Frank Buschmann
Alejandro P. Buchmann
Mariano A. Cilia (Eds.)

LNCS 3013

Object-Oriented Technology

ECOOP 2003 Workshop Reader

ECOOP 2003 Workshops
Darmstadt, Germany, July 2003
Final Reports

Lecture Notes in Computer Science 3013

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Frank Buschmann Alejandro P. Buchmann
Mariano A. Cilia (Eds.)

Object-Oriented Technology

ECOOP 2003 Workshop Reader

ECOOP 2003 Workshops
Darmstadt, Germany, July 21-25, 2003
Final Reports

Springer

Volume Editors

Frank Buschmann
Siemens Corporate Technology, Germany
Otto-Hahn-Ring 6, 81739 Munich, Germany
E-mail: Frank.Buschmann@siemens.com

Alejandro P. Buchmann
Mariano A. Cilia
Darmstadt University of Technology, Germany
FB Informatik, Databases and Distributed Systems
Hochschulstr. 10, 64289 Darmstadt, Germany
E-mail: buchmann@informatik.tu-darmstadt.de
cilia@dvs1.informatik.tu-darmstadt.de

Library of Congress Control Number: 2004108314

CR Subject Classification (1998): D.1-3, H.2, F.3, C.2, K.4, J.1

ISSN 0302-9743
ISBN 3-540-22405-X Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Protago-TeX-Production GmbH
Printed on acid-free paper SPIN: 11299011 06/3142 5 4 3 2 1 0

Preface

This volume represents the seventh edition of the ECOOP Workshop Reader, a compendium of workshop reports from the 17th European Conference on Object-Oriented Programming (ECOOP 2003), held in Darmstadt, Germany, during July 21–25, 2003.

The workshops were held during the first two days of the conference. They cover a wide range of interesting and innovative topics in object-oriented technology and offered the participants an opportunity for interaction and lively discussion. Twenty-one workshops were selected from a total of 24 submissions based on their scientific merit, the actuality of the topic, and their potential for a lively interaction. Unfortunately, one workshop had to be cancelled.

Special thanks are due to the workshop organizers who recorded and summarized the discussions. We would also like to thank all the participants for their presentations and lively contributions to the discussion: they made this volume possible. Last, but not least, we wish to express our appreciation to the members of the organizing committee who put in countless hours setting up and coordinating the workshops.

We hope that this snapshot of current object-oriented technology will prove stimulating to you.

October 2003

Frank Buschmann
Alejandro Buchmann
Mariano Cilia

Organization

ECOOP 2003 was organized by the Software Technology Group, Department of Computer Science, Darmstadt University of Technology under the auspices of AITO (Association Internationale pour les Technologies Objets) in cooperation with ACM SIGPLAN.

The proceedings of the main conference were published as LNCS 2743.

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Workshop Chairs

Frank Buschmann (Siemens Corporate Technology, Germany)
Alejandro Buchmann (Darmstadt University of Technology, Germany)

Table of Contents

Exception Handling in Object Oriented Systems: Towards Emerging Application Areas and New Programming Paradigms 1
Alexander Romanovsky, Christophe Dony, Anand Tripathi, Jørgen Lindskov Knudsen

Parallel Object-Oriented Scientific Computing Today 11
Kei Davis, Jörg Striegnitz

Communication Abstractions for Distributed Systems 17
Antoine Beugnard, Ludger Fiege, Robert Filman, Eric Jul, Salah Sadou

.NET: The Programmer's Perspective 30
Hans-Jürgen Hoffmann

Component-Oriented Programming 34
Jan Bosch, Clemens Szyperski, Wolfgang Weck

Workshop for PhD Students in Object Oriented Programming 50
Pedro J. Clemente, Miguel A. Pérez, Sergio Lujan, Hans Reiser

Formal Techniques for Java-Like Programs 62
Susan Eisenbach, Gary T. Leavens, Peter Müller, Arnd Poetzsch-Heffter, Erik Poll

Object-Oriented Reengineering 72
Serge Demeyer, Stéphane Ducasse, Kim Mens, Adrian Trifu, Rajesh Vasa, Filip Van Rysselberghe

Mobile Object Systems: Resource-Aware Computation 86
Ciarán Bryce, Crzegorz Czajkowski

Quantitative Approaches in Object-Oriented Software Engineering 92
Fernando Brito e Abreu, Mario Piattini, Geert Poels, Houari A. Sahraoui

Composition Languages 107
Markus Lumpe, Jean-Guy Schneider, Bastiaan Schönhage, Markus Bauer, Thomas Genssler

Tools and Environments for Learning Object-Oriented Concepts 119
Isabel Michiels, Jürgen Börstler, Kim B. Bruce, Alejandro Fernández

Patterns in Teaching Software Development 130
Erzsébet Angster, Joseph Bergin, Marianna Sipos

Object-Oriented Language Engineering for the Post-Java Era 143
Wolfgang De Meuter, Stéphane Ducasse, Theo D'Hondt, Ole-Lehrman Madsen

Analysis of Aspect-Oriented Software 154
Jan Hannemann, Ruzanna Chitchyan, Awais Rashid

Modeling Variability for Object-Oriented Product Lines 165
Matthias Riebisch, Detlef Streitferdt, Ilian Pashov

Object Orientation and Web Services 179
Anthony Finkelstein, Winfried Lamerdorf, Frank Leyman, Giacomo Piccinelli, Sanjiva Weerawarana

Advancing the State of the Art in Run-Time Inspection 190
Robert Filman, Katharina Mehner, Michael Haupt

Aliasing, Confinement, and Ownership in Object-Oriented Programming 197
Dave Clarke, Sophia Drossopoulou, James Noble

Author Index 209

Exception Handling in Object Oriented Systems: Towards Emerging Application Areas and New Programming Paradigms

Alexander Romanovsky[1], Christophe Dony[2], Anand Tripathi[3], and Jørgen Lindskov Knudsen[4]

[1] School of Computing Science, University of Newcastle upon Tyne, UK
alexander.romanovsky@ncl.ac.uk
[2] Université Montpellier-II and LIRMM Laboratory, France
dony@lirmm.fr
[3] Department of Computer Science, University of Minnesota, USA
tripathi@cs.umn.edu
[4] Mjølner Informatics A/S, Denmark
jlk@mjolner.dk

Abstract. Exception handling continues to be a challenging problem in object oriented system development. One reason for this is that today's software systems are getting increasingly more complex. Moreover, exception handling is needed in a wide range of emerging application areas, sometimes requiring domain-specific models for handling exceptions. Moreover, new programming paradigms such as pervasive computing, service oriented computing, grid, ambient and mobile computing, web add new dimensions to the existing challenges in this area. The integration of exception handling mechanisms in a design needs to be based on well-founded principles and formal models to deal with the complexities of such systems and to ensure robust and reliable operation. It needs to be pursued at the very start of a design with a clear understanding of the ensuing implications at all stages, ranging from design specification, implementation, operation, maintenance, and evolution. This workshop was structured around the presentation and discussion of the various research issues in this regard to develop a common understanding of the current and future directions of research in this area.

1 Summary of Objectives and Results

There are two trends in the development of modern object oriented systems: they are getting more complex and they have to cope with an increasing number of exceptional situations. The most general way of dealing with these problems is by employing exception handling techniques. Many object oriented mechanisms for handling exceptions have been proposed but there still are serious problems in applying them in practice. These are caused by

- complexity of exception code design and analysis
- not addressing exception handling at the appropriate phases of system development
- lack of methodologies supporting the proper use of exception handling
- not developing specific mechanisms suitable for particular application domains and design paradigms.

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 1–10, 2004

Following the success of ECOOP 2000 workshop[1], this workshop aimed at achieving better understanding of how exceptions should be handled in object oriented (OO) systems, including all aspects of software design and use: novel linguistic mechanisms, design and programming practices, advanced formal methods, etc.

The workshop provided a forum for discussing the unique requirements for exception handling in the existing and emerging applications, including pervasive computing, ambient intelligence, the Internet, e-science, self-repairing systems, collaboration environments. We invited submissions on research in all areas of exception handling related to object oriented systems, in particular: formalisation, distributed and concurrent systems, practical experience, mobile object systems, new paradigms (e.g. object oriented workflows, transactions, multithreaded programs), design patterns and frameworks, practical languages (Java, Ada, Smalltalk, Beta), open software architectures, aspect oriented programming, fault tolerance, component-based technologies.

We encouraged participants to report their experiences of both benefits and obstacles in using exception handling, reporting, practical results in using advanced exception handling models and the best practice in applying exception handling for developing modern applications in the existing practical settings.

The workshop was attended by 18 researchers who participated in the presentation and discussion of ten position papers and one invited talk. These presentations and discussions were grouped into four thematic sessions. The first session (the invited talk and one presentation) addressed engineering systems incorporating exception handling. The focus of the second session (three presentations) was on the specific issues related to object-orientation. In the third session (three presentations) issues related to building novel exception handling mechanisms for distributed and mobile systems were discussed. The topic of the fourth session (three presentations) was exception handling and component based system development.

2 Summary of the Call-for-Papers

The call-for-papers for this workshop emphasized its broad scope and our desire to focus the workshop discussions on problems of perceived complexity of using and understanding exception handling: Why programmers and practitioners often believe that it complicates the system design and analysis? What should be done to improve the situation? Why exception handling is the last mechanism to learn and to use? What is wrong with the current practice and education?

We invited the researchers interested in this workshop to submit their position papers aiming at understanding why exception handling mechanisms proposed and available in earlier OO languages (discussed, for example, at ECOOP 1991 Workshop on Ex-

[1] A. Romanovsky, Ch. Dony, J. L. Knudsen, A. Tripathi. Exception Handling in Object Oriented Systems. In J. Malenfant, S. Moisan, A. Moreira. (Eds.) "Object-Oriented Technology. ECOOP 2000 Workshop Reader". LNCS-1964. pp. 16-31, 2000.

ception Handling and Object-Oriented Programming[2]) are not widely used now. We were interested in papers reporting practical experiences relating both benefits and obstacles in using exception handling, experience in using advanced exception handling models, and the best practices in using exception handling for developing modern applications in existing practical settings.

The original plan was to have up to 20 participants. We asked each participant to present his/her position paper, and discuss its relevance to the workshop and possible connections to work of other attendees. The members of the organizing committee reviewed all submissions. The papers accepted for the workshop sessions were posted on the workshop webpage so the participants were able to review the entire set of papers before attending the workshop.

Additional information can be found on the workshop web page:
http://www.cs.ncl.ac.uk/~alexander.romanovsky/home.formal/ehoos2003.html

The proceedings of the workshop are published as a technical report TR 03-028 by Department of Computer Science, University of Minnesota, Minneapolis, USA:
A. Romanovsky, C. Dony, J. L. Knudsen, A. Tripathi. *Proceedings of the ECOOP 2003 Workshop on Exception Handling in Object-Oriented Systems: Towards Emerging Application Areas and New Programming Paradigms. 2003.*

3 List of the Workshop Presentations

The workshop started with an invited talk delivered by William Bail (Mitre) on *Getting Control of Exception.*

After that following position papers were discussed:

1. Ricardo de Mendonça da Silva, Paulo Asterio de C. Guerra, and Cecília M. F. Rubira (U. Campinas, Brazil). *Component Integration using Composition Contracts with Exception Handling.*

2. Darrell Reimer and Harini Srinivasan (IBM Research, USA). *Analyzing Exception Usage in Large Java Applications.*

3. Peter A. Burh and Roy Krischer (U. Waterloo, Canada). *Bound Exceptions in Object Programming.*

4. Denis Caromel and Alexandre Genoud (INRIA Sophia Antipolis, France). *Non-Functional Exceptions for Distributed and Mobile Objects.*

5. Tom Anderson, Mei Feng, Steve Riddle, and Alexander Romanovsky (U. Newcastle, UK). *Error Recovery for a Boiler System with OTS PID Controller.*

6. Joseph R. Kiniry (U. Nijmegen, Netherlands). *Exceptions in Java and Eiffel: Two Extremes in Exception Design and Application.*

---

[2] Dony, Ch., Purchase, J., Winder. R.: Exception Handling in Object-Oriented Systems. Report on ECOOP '91 Workshop W4. OOPS Messenger **3**, 2 (1992) 17-30

7. Giovanna Di Marzo Serugendo (U. Geneva, Switzerland) and Alexander Romanovsky (U. Newcastle, UK). *Using Exception Handling for Fault-Tolerance in Mobile Coordination-Based Environments.*

8. Frederic Souchon, Christelle Urtado, Sylvain Vauttier (LGI2P Nimes, France), and Christoophe Dony (LIRMM Montpellier, France). *Exception Handling in Component-based Systems: a First Study.*

9. Johannes Siedersleben (SD&M Research, Germany). *Errors and Exceptions – Rights and Responsibilities.*

10. Robert Miller and Anand Tripathi (U. Minnesota, USA). *Primitives and Mechanisms in the Guardian Model for Exception Handling in Distributed Systems.*

4 Summary of Presentations

The first sessions focused on software engineering issues in exception handling. It included two talks. William Bail (Mitre) presented the invited lecture titled *"Getting control of exceptions"*. In his talk he noted that the past developments in this field have allowed programmers to define and use exceptions and this has led a significant advantage in being able to write more reliable software. While not explicitly helping us avoid errors, they enable us to detect their presence and control their effects. Yet they act in opposition to much of what we have learned is good software design - simple structures with well-defined control flows. In addition, they complicate the process of performing formal analyses of systems. This talk elaborated on this issue and projected some potential ideas to help reconcile these challenges, especially with the use of OO concepts. The second talk in the first session was given by Johannes Siedersleben (SD&M Research, Germany). He presented the paper *"Errors and Exceptions - Rights and Responsibilities"*. The talk emphasized the strict separation of errors to be handled by the application and the 'true' exceptions which require recovering and restart mechanisms. It suggested the use of the term "emergency" for the exceptions of the second type because in many programming languages, exceptions can and are used for many non-exceptional situations. The paper also describes a component-based strategy to handle emergencies using so called safety facades.

The theme of the second session centered on exception handling in OO Systems. It included three papers. The first talk in this session was by Harini Srinivasan (IBM Research, USA), who presented the paper "Analyzing Exception Usage in Large Java Applications". This talk emphasized that proper exception usage is necessary to minimize time from problem appearance to problem isolation and diagnosis. It discusses some common trends in the use of exceptions in large Java applications that make servicing and maintaining these long running applications extremely tedious. The talk also proposes some solutions to avoid or correct these misuses of exceptions. The second presentation was by Roy Krischer (U. Waterloo, Canada) on the paper entitled *"Bound Exceptions in Object Programming"*. Many modern object-oriented languages do not incorporate exception handling within the object execution environment. Specifically, no provision is made to involve the object raising an exception in the catching mechanism in order to allow discrimination among multiple objects raising the same exception. The notion of bound exceptions is introduced, which as-

sociates a 'responsible' object with an exception during propagation and allows the catch clause to match on both the responsible object and exception. Multiple strategies for determining the responsible object were discussed in this talk, along with extending bound exceptions to resumption and non-local propagation among coroutines/tasks. The third speaker in this session was Joseph R. Kiniry (U. Nijmegen, Netherlands) who presented his paper *"Exceptions in Java and Eiffel: Two Extremes in Exception Design and Application"*. His focus was on an analysing the exception handling mechanisms in the Java and Eiffel languages and on contrasting the style and the semantics of exceptions in these two languages. The talk showed how the exception semantics impacts programming (technically) and programmers (socially). According to the author the primary result of this analysis is that Java checked exceptions are technically adequately designed but are socially a complete failure. This position is supported by an analysis of hundreds of thousands of lines of Java and Eiffel code.

The third session had three talks on exception handling in mobile and distributed systems. Alexandre Genoud (INRIA Sophia Antipolis, France) presented the paper *"Non-Functional Exceptions for Distributed and Mobile Objects"*. He proposed the notion of non-functional exceptions to signal failures occuring in non functional properties (distribution, transaction, security, etc.). He described a hierarchical model based on mobile exception handlers. Such handlers, attached to distribution-specific entities (proxies, futures), are used to create middleware-oriented handling strategies. The handling of exceptions can indifferently be at non-functional or application-level. The second speaker in this session was Alexander Romanovsky (U. Newcastle upon Tyne, UK), who presented the paper *"Using Exception Handling for Fault-Tolerance in Mobile Coordination-Based Environments"*. Mobile agent-based applications very often run on a mobile coordination-based environment, where programs communicate asynchronously through a shared memory space. The aim of this paper is to propose an exception handling model suitable for such environments. It is our view that it is conceptually wrong to treat such exceptions as usual events or tuples. This is why in the model proposed a local handler agent is created each time when an exception is signalled: this guarantees handling, allows exceptions and handlers to be dynamically associated and decreases the overall overheads. The third talk in this session was presented by Anand Tripathi (University of Minnesota, Minneapolis, USA) on *"Primitives and Mechanisms in the Guardian Model for Exception Handling in Distributed Systems."* In this talk he elaborated on notion of the guardian for encapsulating exception handling policies in a distributed application. He presented the core set of primitives of the guardian model which allow the programmer to specify and control the recovery actions of cooperating processes so that each process performs the required exception handling functions in the right context. This talk elaborated on how the various other existing models for distributed exception handling can be implemented using the guardian model.

The fourth session in the workshop focused on exception handling issues related to software and systems composition. Paulo Asterio de C. Guerra (U. Campinas, Brazil), presented the paper *"Component Integration using Composition Contracts with Exception Handling"*. He outlined an architectural solution for the development of dependable software systems out of concurrent autonomous component-systems. The solution is based on the concepts of coordination contracts and Coordinated Atomic (CA) Actions, which are adapted to a service-oriented approach. The second talk in

this session was by Mei Feng (U. Newcastle upon Tyne, UK) on the paper *"Error Recovery for a Boiler System with OTS PID Controller"*. The talk presented the protective wrapper development for the model of the system in such a way that they allow detection and tolerance of typical errors caused by unavailability of signals, violations of range limitations, and oscillations. In the presentation the case study demonstrated how forward error recovery based on exception handling can be systematically incorporated at the level of the protective wrappers. The last talk of this session was given by Christelle Urtadeo (LGI2P – Ecole des Mines d'Ales, Nimes, France) on *"Exception Handling in Component Based Systems: A First Study.* Christelle Urtado presented a preliminary study on exception handling in component-based systems written by F. Souchon, C. Urtado, S. Vauttier and C. Dony. The talk focused on the category of components that interact in a contract-based manner and communicate asynchronously. According to the authors, exception handling for such components should provide four features: handler contextualization, concurrent activity coordination, exception concertation and exception handling support for broadcasted requests. These requirements have already shown to be pertinent in a similar context: the SaGE exception handling system that has been designed by the authors for multi-agent systems. The speaker has used SaGE implementation to exemplify how exception handling should be managed for the considered components.

5 Summary of Discussions

The workshop has gathered a rich collection of contributions covering many of the subjects that constitute the exception handling and fault tolerance domains. Presentations had generated numerous questions and exchanges. The main goal of the concluding 45 minutes general discussion was to bring to the fore the main conclusions, results, challenging issues and research directions implicitly or explicitly expressed during papers presentations and discussions on the four main subjects addressed during the workshop:

- Software engineering issues in exception handling,
- Exception handling issues in today's standard object-oriented systems
- Reliable mobile, distributed, concurrent systems
- Reliable component-based systems.

Discussion on the first issue led the attendees to a primary and major conclusion that after almost thirty years of research and many years of experimentation with many different languages, it appears that our knowledge of language primitives for exception handling is somehow high but that there is a huge need for standard definitions and for standard analysis, design and programming patterns. Indeed, the primitive for exception handling in today's languages are, in one way or another, evolutions of those proposed in the seminal paper by John Goodenough written in1975. They are primitives for signaling, catching, and handling of exception based different execution models such as entailing termination (or retry), resignaling (or propagation), or resumption. As far as these crucial concepts have been correctly extracted from the foundational research, important progress have been made in their understanding, adaptation, development and implementation. Future research efforts will need to

adapt these primitives to tomorrow's needs. Unfortunately, there has never been a basic agreement, a norm, on the definition of the terms "exception", "exception handling", "fault tolerance".

Early terms such as "domain", "range" or "monitoring" exceptions as proposed in Goodenough's paper are not standardised mainly because they do not reflect the concept complexity. Researchers thus have to choose or to re-invent ever again their own definitions : consider the following terms, all coming from previous works, which have been used in our workshop contributions to denote either the same thing or subtly different things : exception, error, warning, condition, alarm, emergency, etc. There is neither a general agreement on the standard patterns to handle exceptions or to write fault-tolerant or defensive programs. When architectural solutions exist, they are not known of today's developers because there is no reference book where such patterns are described. It is interesting to note that this issue was quoted as an open issue in our ECOOP 2000 workshop conclusion, and that no significant advance has been made in that direction. However, this year, new interesting papers have reported experiences on how developers, either experienced or not, deal with exceptions, how they sometimes reinvent known solutions and make known mistakes, how they sometimes misuse or misunderstand language constructs. Everybody has thus agreed to stress the need for dedicated design patterns and to present this as a major issue.

The second main point in the discussion concerned the use of exception handling techniques in today's object-oriented systems in general and more particularly in Java as far as this language has been at the heart of the debates. Java is today "de facto" a vast field of experimentation for exception handling because it makes it mandatory for programmers to deal with exceptions, especially for the so-called "checked exceptions" (see section 4). Dealing with exception at a large scale, as experienced for example by Ada developers, had never been done by object-oriented developers because no language before Java mandated it. As William Bail noticed in the workshop introductory talk, the history of exceptions is a love/hate relationship and programmers generally do not like to handle exception for various good or wrong reasons: it makes programs longer to write, breaks code harmony, is boring, seems useless, or seems to slow execution time down.

The discussion thus focused on issues connected to Java design choices and constraints (e.g. static typing) related to exception handling. Some of these issues are already well known and sometimes related solutions exists and have been published years ago; they have however been considered here in the light of new experience reports. A first one is that handling (putting the system back into a coherent state within catch clauses body) cannot be performed in a generic way by sending messages to the exception object. Examples have been presented showing how it can impose to write several catch clauses where one would be enough instead. Besides, the exception objects do not contain enough information.

More generally, and William Bail also indirectly quoted this in his introductory talk, Java's exception handling system is certainly and for many reasons not enough object-oriented. A connected problem is that a *try block* with an empty *catch clauses* (entailing termination) can be understood by beginners, or used by developers in a

final project stage, as doing nothing else than magically suppressing compiler errors. This remark introduced the main point of the debate on checked and unchecked exceptions. The question has been raised to know whether to force programmers to trap checked exceptions or to declare them in methods signature is not the cause of many misuses of exception handling in Java. For example, the systematic use of empty catch clauses has been itself reported as a major reason of debugging difficulties in some late large-scale projects. We have certainly not given the final answer to that question. The idea of checked exceptions, introduced by Liskov and Snyder in CLU, seems in itself very coherent, rigorous and fruitful but Java's experience tends to show that it has initially unpredicted consequences. As far as such control on software code is wanted, the alternative solution proposed in Beta to handle with different mechanisms, one static and one dynamic, failures (conceptual equivalent of Java's unchecked exceptions) and exceptions (checked ones) should be considered. The evolution of exception handling models for standard OO programming is still an open issue.

The ECOOP 2000 workshop conclusion stressed the need for new exception handling linguistic mechanisms and patterns that take into account new or difficult programming paradigms used to develop mobile, distributed, concurrent or component based systems. Discussions on those points began by stating that workshop contributions revealed significant advances, primarily for what concerns distributed systems where several deep and complete studies have been achieved. However, many open issues and problems require further investigation into practical applications of some of the emerging models. The most fundamental issues with exception handling in distributed and concurrent systems are related to policies and mechanisms for resolution of concurrent exceptions, determining which exceptions should be signaled in a process that is required to participate in recovery, and the context in which a process should handle the exception delivered to it. This requires models and mechanisms to ensure that each process handles a global exception in the proper context. There are several existing models that facilitate this by imposing certain program structuring disciplines such as conversations or transactions. The Guardian model can be considered as a basic model, addressing some of the basic issues to serve as meta-level model that can be used to implement other models, including the existing ones. However, more detailed and practical studies are needed in this area.

Mobile and component-based systems and applications raise newer, difficult and interesting issues for which propositions or state-of-the-problem have been described. The primary issues arise due to a broad class of exceptions that can arise due to distribution, asynchronous interactions among components, and due to a relatively large number of unanticipated conditions in which the component of an application could be used. Mobile agents represent one class of components, which are active (objects) that are capable of migrating from one execution environment to another. A mobile agent (object/component) executing in a remote environment" may not be able to determine and handle the context of an exception when situated in a remote environment. A number of models and approaches were discussed in this workshop. These include separation between functional and non-functional exceptions, and support for exception handling systems with asynchronous communication based on tuple spaces.

Component programming models extend object-oriented programming and to correctly integrate exception handling and fault tolerance in the design and implementation of new component languages raise issues at all stages. If existing propositions are to be considered, those issues also are as different as today's models. Component can for example be distinguished by the kind of interfaces they propose, the way they interact (e.g. contract-based or event-based interactions) and the way they communicate (e.g. synchronous or asynchronous communications). Some hard point brought to the fore by the discussions, for which extended research efforts will certainly be needed, are for example validation of composites made from components able to signal exceptions or the control of asynchronous distributed components such as Java's message-driven beans. Component based software engineering methods are going to be central in building pervasive computing systems and "smart environments" whose intrinsic model of computation and user-interactions is built upon supporting dynamic discovery of services/components and their interactions with mobile user devices. Fault-tolerant and safe operations of such systems will require addressing of exceptions handling issues of even greater complexities than those being encountered and addressed in today's systems.

6 Conclusions

Exception handling systems are designed to enhance software reliability, reusability, readability and debugging. During the 1980s and 1990s, exception handling systems were integrated into all object-oriented languages.

The first ECOOP workshop on exception handling and OO programming held in 1991 was oriented towards the specification, understanding and implementation of the first versions of object-oriented exception handling systems which can be classified into various families. The Smalltalk or Clos family which proposed dynamically-typed, very powerful in term of expressive power, fully object-oriented, systems represented in fine by the ANSI Smalltalk Standard in 1997. The C++ and Java family, representing the 1995-2005 tendency, with less expressive power and various restrictions which are either design choices (e.g. Checked exceptions) or consequences of the quest for static and correct typing (e.g. restrictions on interface specialization). Eiffel or Beta proposed important variations on exception handling in the same context of statically-typed languages.

The second ECOOP workshop on exception handling, held in 2000 in Nice, dealt less with standard languages implementation, it brought to the fore a large panel of open and challenging problems: models for standard languages are were not entirely satisfactory, many important past contributions were forgotten in standard languages, there were a well identified set of research works advocated towards (1) exception handling systems safe and compatible with static and correct typing, (2) new linguistic mechanisms to take into account concurrent, web, mobile, distributed systems, (3) the need for disciplined exception handling at all phases of system development and (4) the need for good development techniques with well-developed architectural and design patterns.

This workshop, three years later, has been a clear success. The introduction talk ideally put the things in situation by recalling basic knowledge and challenges. The range of addressed topics has been wide and the technical level of presentations and discussions high, either for what concerns research contributions or experience reports.

1. For the first time we have had detailed and precise developers experience reports on exception handling in object-oriented languages. They have clearly shown us on the one hand that some of Java design choices or typing constraints really induce problematic programming practice that should influence the design of future versions or of future standard languages. On another hand, they also clearly show that there is a huge need, as quoted in 2000, for a book on exception handling design patterns. This now is a crucial point for an actual development of fault tolerant programming.
2. Advances, sometimes very important, have been made in some of the problem quoted as challenging in 2000. This is the case for exception handling in distributed, concurrent, asynchronous-communication based or mobile systems. New ideas have been proposed on various fields as e.g. a new original language-level basic mechanisms (so-called bound exceptions, see section 4).
3. Finally, many issues remain or have appeared. (1) For example, as far as exception handling is not just a linguistic issue, and we still do not have many models and tools to deal with reliability at all stage of software development. (2) Today's tendency still is to use statically-typed OO languages, albeit other voice still can be heard, but problems related to static-typing are not solved in Java, languages that propose to combine inclusion polymorphism and contravariance (Beta) are not studied enough and multiple dispatch is either too slow or is too space-consuming. (3) The separation of concerns in the case of exception handling would be a great advance but a ECOOP 2000 paper has shown that it was a difficult and controversial issue. Much more work should be put on that point. (4) The representation of exceptions by hierarchically organized classes is now quite a standard but the design issues related to that representation are still not solved and design languages such as UML do not correctly take it into account. There is a lack of proper frameworks and principles for designing and analyzing exception hierarchies and exception code in large-scale systems.
4. New difficult issues, connected to the development of new technologies, have been brought to the fore, especially for what concerns component-based systems. It has been shown, for example, that current component systems architectures exception handling systems are quite poor and limited.

This workshop has really been a positive event, and it has reinforced everyone's view that such a meeting is of a tremendous value to the reliable software development and research community involved in object-oriented and component-based development.

Acknowledgements. We would like to thank the participants for making this workshop a success with their unique contributions. This report reflects the viewpoints and ideas that were contributed and debated by all these participants. Without their contributions, neither the workshop nor this report would have materialized.

Parallel Object-Oriented Scientific Computing Today

Kei Davis[1] and Jörg Striegnitz[2]

[1] Algorithms, Modeling, and Informatics Group CCS-3, Los Alamos National Laboratory, Los Alamos, NM 87545, USA,
Kei.Davis@lanl.gov, http://www.c3.lanl.gov/
[2] Forschungszentrum Jülich GmbH
Central Institute for Applied Mathematics (ZAM)
52425 Jülich, Germany
J.Striegnitz@fz-juelich.de

High-performance scientific computing is arguably of greater importance than heretofore in the history of computing. Traditionally used primarily in support of fission- and fusion bomb design and weather forecasting, numerically-intensive applications running on massively parallel machines may be found modeling any number of natural phenomena from the smallest scale to the largest—from nuclear dynamics to the formation and evolution of the universt—as well as for the design of human artifacts—automobiles, aircraft, internal combustion engines, pharmaceuticals—the list seems endless. With the announcement of the Japanese Earth Simulator—a machine ostensibly designed expressly for global environmental modeling—and subsequent announcements by other institutions and governments of intents to build comparable machines for similar purposes, it is arguable that such simulations will be the primary consumer of supercomputing capacity worldwide.

Arguably too, more than ever hardware capabilities exceed practioners' ability to utilize them to their fullest potential. Though state-of-the-art architectures may be short of ideal (an ideal yet to be discovered), it is clear in perspective that the current tools and methodologies for designing and implementing these applications are primitive. Indeed, the state of practice in high-performance scientific computing notably lags the rest of the computing world: consider that FORTRAN 77, FORTRAN 90, and C are the programming languages still in greatest use.

Why has this field, with the largest, most highly parallel, and probably most complex applications, not embraced the newest and best of current practice, much less contributed significantly to research into paradigms for design and implementation yet more appropriate and effective than the best of general practice? There are several plausible answers to this. The simplest is simply the inertia of entrenched practice and the volume of existing (legacy) code. A second is a consequence of formal training and so mindset—the majority of the practitioners are not computing scientists or software engineers but physicists, mathematicians, and engineers with neither appreciation for or practical knowledge in computing science and software engineering as legitimate and useful disciplines, nor the time or inclination to rectify this lack—after all, they have

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 11–16, 2004.

their own disciplines to pursue. A third reason, sadly, reflects a political reality: the largest-scale computing facilities and their staff are typically government funded, with the usual short-term (and so short-sighted) funding horizons and corresponding expectations of immediate concrete 'results.'

In spite of all this a number of researchers and research groups worldwide are working assiduously, even valiantly, to advance the state of the art of high-performance scientific computing. A first step—seemingly cautious to the outsider but rather bold in their environment—is the adoption and adaptation of the best of proven current design and programming practice. At this point in time this seems to be the use of the object-oriented paradigm as a successor to the procedural. We shall not attempt to extol the advantages of this approach here, but simply point out that the new abstraction mechanism provided yields a benefit not realized by more mainstream computing: the ability to 'abstract away' (hide the complex details of) parallel execution (and concomitant interprocess communication) and the distribution of data.

At the inception of this new field the single venue specific to relevant research was ISCOPE—the International Symposium on Computing in Object-Oriented Parallel environments. In time, however, this has transmogrified into what is essentially a forum for Java language issues, as evidenced by its being largely eclipsed and absorbed by the JavaGrande conference. To maintain focus on what is demonstrably still an active area of research an intermittent sequence of these POOSC workshops have been held. The premier European Conference on Object-Oriented Computing (ECOOP), with its rather more broad ambit that the American counterpart Object-Oriented Progamming, Systems, Languages, and Applications (OOPSLA), has proven to be an excellent umbrella for this activity.

1 The Subject Matters in Brief

The complete set of papers is available as *Proceedings of the Workshop on Parellel/high-Performance Object-Oriented Scientific Computing (POOSC'03)*, Jörg Striegnitz and Kei Davis (eds.), Technical Report FZJ-ZAM-IB-2003-09, July 2003, Forschungszentrum Jülich GmbH, Zentralinstitut für Angewandte Mathematic (ZAM), D-52425 Jülich, Germany. Following is a brief discussion of each and the contexts in which they were engendered.

1.1 OO at All?

While object-oriented programming is being embraced in industry, particularly in the form of C++ and to an increasing extent Java, its acceptance by the parallel scientific programming community is still tentative. In this latter domain performance is invariably of paramount importance, where even the transition from FORTRAN 77 to C is incomplete, primarily because of performance real or perceived loss. On the other hand, three factors together practically dictate the

use of language features that provide better paradigms for abstraction: increasingly complex numerical algorithms, application requirements, and hardware (e.g. deep memory hierarchies, numbers of processors, communication and I/O).

In *To OO or not to OO* Manuel Kessler addresses in more detail the issues affecting acceptance of object orientation in high-performance computing.

> The High Performance Computing (HPC) community lags between 10 and 20 years behind other Information Technology (IT) areas. We figure out some of the reasons for this sluggish acceptance and incorporation of new technologies and try to find some arguments and prerequisites for a more lively and finally successful embracement of new insights, in order to be able to tackle the challenges of future HPC applications. However, OO—as any other technology—is no silver bullet, and some care is indicated.
>
> Object-Oriented Design—and even more so OO Implementation—is still a new and for some people even suspicious approach to software development in the HPC field. The reasons for this unfortunate situation are many, among them
>
> – real performance loss;
> – perceived performance loss (rumor, outdated experience);
> – lack of training;
> – lack of time;
> – complexity of OO languages, like C++ or Java;
> – plain laziness, or, less offensive, inertia.

1.2 The Hard Core

Approximately half of the presentations were on topics directly concerned with particular HPC application areas.

In *Object-Oriented Framework for Multi-method Parallel PDE Software,* Pieter De Ceuninck et al. described a study of the application of object oriented (OO) techniques in the design of parallel software for the numerical solution of partial differential equations. In *Object-Oriented and Parallel Library for Natural Convection in Enclosures,* Luis M. de la Cruz Salas et al. presented advances in the construction of class libraries for solving the governing equations of natural convection in enclosures, concentrating on newtonian and incompressible fluids. Guntram Berti, in *Generic Programming Support for Mesh Partitioning Based Parallelization,* presented a domain-specific, library based approach for parallelizing mesh-based data-parallel applications (like numerical PDE solution) using the domain partitioning paradigm. Finally, Frank Schimmel in *An Object-Oriented Design for an Atmospheric Flow Model* provided a case study in OO design, with emphasis on patterns and their modifications to accommodate the special needs of a scientific computing application, e.g. to improving performance.

1.3 Tools, Techniques, and Enabling Technologies

In *PEDPI as a Message Passing Interface with OO Support,* Zoltán Hernyák described a parallel event-driven programming interface, developed in the Microsoft .NET environment, that supports object-oriented parallel program development. Alexander Linke et al., in *Fast Expression Templates for the Hitachi SR8000 Supercomputer*, demonstrate a novel exploitation of C++ expression templates for scientific computing for improving on the performance of 'classical' expression templates. In *Dealing with Efficiency and Algebraic Abstraction in Vaucanson,* Yann Régis-Gianas and Raphaël Poss describe a new design pattern they call ELEMENT which enables the orthogonal specialization of generic algorithms with respect to the underlying algebraic concept and and its implementation. The essential idea is to make concept and implementation explicitly usable as object instances. Finally, recalling the the functional programming paradigm supports OO programming and is eminently suited to the expression of parallel scientific computations, Zoltán Horváth et al., in *Functional Programs on Clusters,* describe implemented Clean-CORBA and Haskell-CORBA interfaces as a way for developing parallel and distributed applications on clusters consisting of components written in functional programming languages such as Clean and Haskell.

2 Authors and Institutions

Guntram Berti, C&C Research Laboratories, NEC Europe Ltd. Rathausallee 10, 53757 St. Augustin, Germany.

Pieter De Ceuninck, Dept of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium.

Herman Deconinck, von Karman Institute for Fluid Dynamics, Waterloosesteenweg 72, B-1640 Sint-Genesius-Rode, Belgium.

Alexander Linke, Universität Würzburg, Institut für Angewandte Mathematik und Statistik, Am Hubland, D-97074 Würzburg, Germany.

Zoltán Hernyák, Eszterházy Károly College, Department of Information Technology, 1. sqr. Eszterházy , Eger 3300 Hungary.

Zoltán Horváth, Department of General Computer Science, University of Eötvös Loránd, Budapest, Hungary.

Manuel Kessler, Institute for Aerodynamics and Gasdynamics (IAG), University of Stuttgart, Pfaffenwaldring 21, D-70550 Stuttgart, Germany.

Eduardo Ramos Mora, Energy Research Centre, UNAM, Mexico.

Alexander Pflaum, Universität Würzburg, Institut für Angewandte Mathematik und Statistik, Am Hubland, D-97074 Würzburg, Germany.

Tiago Quintino, Dept of ComputerScience, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium and von Karman Institute for Fluid Dynamics, Waterloosesteenweg 72, B-1640 Sint-Genesius-Rode, Belgium.

Yann Régis-Gianas, LRDE: EPITA Research and Development Laboratory, 14-16, rue Voltaire - F-94276 Le Kremlin-Bicêtre Cedex, France.

Raphaël Poss, LRDE: EPITA Research and Development Laboratory, 14-16, rue Voltaire - F-94276 Le Kremlin-Bicêtre Cedex, France.

Luis M. de la Cruz Salas, Applied Computing, DCI, DGSCA-UNAM, Mexico.

Frank Schimmel, Meteorologisches Insitut, Universität Hamburg, Germany.

Stefan Vandewalle, Dept of ComputerScience, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium.

Zoltán Varga, Department of General Computer Science, University of Eötvös Loránd, Budapest, Hungary.

Viktória Zsók, Department of General Computer Science, University of Eötvös Loránd, Budapest, Hungary.

3 Organizers and Committee

Dr. Kei Davis – Co-chair
Modeling, Algorithms, and Informatics, CCS-3 MS B256
Los Alamos National Laboratory
Los Alamos, NM 87545, U.S.A.
Kei.Davis@lanl.gov

Jörg Striegnitz – Co-chair
Forschungszentrum Jülich GmbH
Central Institute for Applied Mathematics (ZAM)
52425 Jülich, Germany
J.Striegnitz@fz-juelich.de

Prof. Vladimir Getov
University of Westminster
School of Computer Science, University of Westminster
Watford Rd, Northwick Park
Harrow, London HA1 3TP, UK
V.S.Getov@wmin.ac.uk

Prof. Andrew Lumsdaine
Computer Science Department
Indiana University
215 Lindley Hall
Bloomington, IN 47405, U.S.A.
lums@cs.indiana.edu

Dr. Bernd Mohr
Central Institute for Applied Mathematics (ZAM)
Forschungszentrum Jülich GmbH
52425 Jülich, Germany
B.Mohr@fz-juelich.de

Dr.-Ing. Jörg Nolte
FhG-FIRST
Kekuléstraße 7
12489 Berlin, Germany
Joerg.Nolte@first.fraunhofer.de

Acknowledgments. We thank all of the contributors, referees, attendees, and the ECOOP workshop organizers for helping to make this workshop, once again, a highly successful event.

Communication Abstractions for Distributed Systems

Antoine Beugnard[1], Ludger Fiege[2], Robert Filman[3], Eric Jul[4], and Salah Sadou[5]

[1] ENST-Bretagne, Brest, France, antoine.beugnard@enst-bretagne.fr
[2] Darmstadt University of Technology, Germany fiege@gkec.tu-darmstadt.de
[3] RIACS/NASA Ames Research Center, USA rfilman@mail.arc.nasa.gov
[4] University of Copenhagen, Copenhagen, Denmark eric@diku.dk
[5] Université de Bretagne Sud, Vannes, France sadou@iu-vannes.fr

Abstract. Communication is the foundation of many systems. Understanding communication is a key to building a better understanding of the interaction of software entities such as objects, components, and aspects. This workshop was an opportunity to exchange points of view on many facets of communication and interaction. The workshop was divided in two parts: the first dedicated to the presentation of eight position papers, and the second to the selection and discussion of three critical topics in the communication abstraction domain.

1 Introduction

As applications become increasingly distributed and as networks provide more connection facilities, communication takes an increasingly central role in modern software systems. Many different techniques and concepts have been proposed, in both research and industry, for providing structure to the problems of communication in software systems. Over the last 15 years, the basic building blocks for distributed object systems have emerged: distributed objects, communicating with Remote Message Send (RMS), also known as Remote Method Invocation (RMI) or Location-Independent Invocation (LII). Message-oriented middleware has also seen some clever implementations. However, it has also become clear that while such abstractions are by themselves sufficient to expose the hard problems of distributed computing, they do not solve them.

Databases and graphical user interfaces are examples of software elements that were once difficult to program but have, through the right abstractions and implementations, been greatly simplified. The question central to this workshop is whether some similar abstractions and can be devised for simplifying distributed applications. Just as successful database components required not only mechanisms for storing and retrieving data, but also abstractions like query languages and transactions, successful communication abstractions will require more than the mere ability to communicate.

At previous ECOOP workshops on *Communication Abstractions*, we identified some features (security, reliability, quality of service, run-time evolution,

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 17–29, 2004.

causality) that are central to any communication abstraction and some particular communication abstractions such as Peer-to-Peer or Publish/Subscribe. The goal of this workshop was to work on the definition of new and better communication abstractions and on the distributed-specific features mentioned above.

We received 17 positions papers. Three were related to cryptography only and were considered out of scope. All others were reviewed by at least two members of the program committee and 8 were considered bringing an interesting point of view and deserving a chance to be discussed.

We organized the workshop as follows:

- The morning was dedicated to short, 15 minute presentations of selected papers. The paper authors entertained questions from the workshop attendees and provided clarifying responses.
- In the afternoon, we formed three working groups for deeper discussion of particular issues in communication abstractions. The group reported their conclusions to the collected workshop at the end of the day.

2 Position Papers, Abstracts, and Discussions

2.1 Event Based Systems as Adaptative Middleware Platforms [7]

Adaptive middleware is increasingly being used to provide applications with the ability to adapt to changes such as software evolution, fault tolerance, autonomic behavior, or mobility. It is only by supporting adaptation to such changes that these applications will become truly dependable. In this paper we discuss the use of event based systems as a platform for developing adaptive middleware. Events have the advantage of supporting loosely coupled architectures, which raises the possibility of orthogonally extending applications with the ability to communicate through events. We then use this ability as a way to change the behavior of applications at run time in order to implement the required adaptations. In the paper we briefly describe the mechanisms underlying our approach and show how the resulting system provides a very flexible and powerful platform in a wide range of adaptation scenarios.

Questions and Answers

How do you know where to put the hooks on the application? We assume we have access to the source code of the applications and know how to generate the point cuts that are needed. In some cases it is not necessary to have access to the source code as we can use PROSE to trap generic operations like outgoing socket connection calls to a certain location. Nevertheless, we are right now assuming that the source code or at least the relevant parts of the source code are known.

Centralized vs. distributed. We currently use an approach similar to Bluetooth and we pre-allocate the master. We have not yet done performance studies on the differences between centralized and distributed event management.

Only application extensions or middleware extensions as well? We can do both. Certainly the application. The event system must run in Java and be amenable to PROSE but in principle it is possible to extend the event system as well.

What about the Reliability model? Right now we do not have reliability. We were trying to prove the concept of dynamic adaptability using events and we have not yet gone as far as considering reliability. At this stage we are focused on basic performance problems inherent to wireless networks. Once those are solved, we will then add additional layers of functionality such as reliability, security, etc.

Without reliability is the system usable? No. But what we have is only the first step. We have not yet completed the system and there are several layers missing to make it a realistic platform. Reliability will be implemented at a higher level than what has been described in the talk.

Removing and introducing new functionality, cross interactions. We have two type of extensions, a non-aspect extensions (module replacement) and an aspect based extension that supports further extensions. The assumption is that the person introduces extensions knows what she is doing. Extensions are part of the code of an application and one can create bugs much as bugs are introduced in normal code.

What devices do you use? Laptops in a first attempt. IPAQs with Linux and Java right now.

How do you map new events to the parameters and the actual application? PROSE uses the JVMDI to extract the parameters of calls and we use that information to cast parameters as necessary. For matching parameters, we use a simple algorithm that simply does pattern matching. Future versions will do a more complete job in this area.

2.2 Interaction Systems Design and the Protocol and Middleware Centered Paradigms in Distributed Application Development [2]

This paper aims at demonstrating the benefits and importance of interaction systems design in the development of distributed applications. We position interaction systems design with respect to two paradigms that have influenced the design of distributed applications: the middleware centered and the protocol-centered paradigm. We argue that interaction systems that support application-level interactions should be explicitly designed, using the externally observable behavior of the interaction system as a starting point in interaction systems design. This practice has two main benefits: to promote a systematic design method, in which the correctness of the design of an interaction system can be assessed against its service specification; and, to shield the design of application parts that use the interaction system from choices in the design of the supporting interaction system.

Questions and Answers

What is the notion of platform-independence you adopt? The application part is defined at a high level of platform independence, which means that the applications parts are defined in a way that is unconstrained by potential target platforms. This is because they are defined in terms of the external behaviour of the interaction system.

The design of the interaction system itself may rely on an abstract platform, which will define the notion of platform-independence explicitly, by defining what characteristics of potential platforms are to be considered.

Can there multiple interaction systems coexist? What if they interact? Yes, they can coexist. There is a duality between interaction systems and system parts. If a system part interacts with two or more interaction parts, it can be considered itself an interaction system.

How do you compare your approach with approaches in coordination languages, EAI or MDA? This comparison is the next step of this approach. We have a set of design concepts that have been derived from LOTOS. The group I work on was involved in the development of LOTOS and departed from it for a more expressive set of basic design concepts based on interactions and relations between interactions. As future work, we should look at other coordination languages and see how the basic concepts can be mapped[1].

Is the boundary between two systems is itself an interaction system? Yes, and it should be considered explicitly as a interaction system if it is interesting according to the criteria I've presented for justifying interaction system design. This is possible in the approach because of the recursive definition of interactions systems.

Where does one stop? (Because otherwise you'll have infinite interaction systems.) We should stop when the interactions are simple enough to be realized at implementation. This is arbitrary, again according to the criteria to justify interaction system design.

How does the boundary between system parts look like? How to define it? We have a set of basic design concepts for that. Interaction points are abstractions of the mechanisms shared by two system parts for interworking. Interactions occur at interaction points. Interactions are shared activities. They may take time and occur synchronously, in that all system parts involved in the interaction perceive the occurrence at the same time when the interaction completes (and have the same perspective on that). The use of synchronous interactions allow us to model tightly coupled interactions at a very high level of abstraction (without considering implementations of the synchronous interactions)[2].

Do system parts have to know (be aware of) the interaction points? This is a basic set of design concepts that allows a mapping to different implementations solutions. The interaction points is an abstraction of the mechanisms shared by two system parts.

[1] with respect to the relation to MDA please have a look at [10]

[2] please have a look at reference [1]

2.3 A Modular QoS-Enabled Load Management Framework for Component-Based Middleware [4]

Services are increasingly dependent on distributed computer systems for office workstation support, banking transactions, supermarket stock supply, mobile phones, web services etc. Web services are being increasingly used for e-commerce, education and for information access within and between organizations. This leads to increasing need for high performance, enterprise level, distributed systems. The most suitable way to achieve high performance is by using load management systems. Delivering end-to-end QoS for diverse classes of applications continues to be a significant research area. There are individual technologies, based on prior research, generally targeting only specific domains and usage patterns. This paper presents a new QoS-enabled load management framework for component oriented middleware. It proposes a new approach for load management and for delivering end-to-end QoS services. The proposed framework offers the possibility of selecting the optimal load distribution algorithms and changing the load metrics at runtime. The QoS service level agreements are made at user level, the application being managed not being aware of this. Work is in progress for creating a simulation model for the proposed framework and for evaluating the performance improvements it offers, with the current focus on the enterprize java beans platform.

2.4 Communication Abstractions in MundoCore [6]

Ubiquitous computing with its many different devices, operating system platforms and network technologies, poses new challenges to communication middleware. Mobile devices providing vastly different capabilities need to integrate into heterogeneous environments. We have identified a key set of requirements for ubiquitous computing communication middleware which are modularity, small footprint, and ability to cope with handovers. In this paper we present MundoCore which provides a set of communication abstractions based on Publish/Subscribe for ubiquitous computing scenarios. We present the communication model of MundoCore and discuss our implementation, along with our experiences and observations from the implementation process.

Questions and Answers

What is currently implemented? MundoCore on C++ and Java. IP-based transport services with optional AES-128 encryption. Routing service. Event routing service. Remote method calls based on own precompiler. Language extensions based on own precompiler. Current Limitations: Currently only SOAP message format supported, Event filters simplistic, No access services (DUN)

Could it be ported to an Ericson Smartphone? Yes. The footprint of the C++ version is below 100KB. We plan to do a port on Nokia Series 60 (EPOC) in the near future.

How does 'emits' behave in case of inheritance? Emits is implemented by means of protected static inner classes. If the enclosing class is derived, the emitter-class is also derived. This way, inheritance is fully supported by using standard Java language features.

2.5 CSCWGroup: A Group Communication System for Cooperative Work in the WAN [9]

Group communication is an important part of CSCW system. Originally its studies focused on LAN environment, and later on WAN for users concentratedly in several areas. However, it was rarely discussed for group members distributing dispersedly. For widely supporting the group communication in the WAN, an new group communication system– CSCWGroup is built. The client/server cluster architecture is introduced in our system. Based on it, a set of management mechanisms is designed in this paper. For evaluating the system performance, simulation software JAVASIM is used to simulate the running process of the system. With the measurement results of message delay and system throughput, it is proven that the system has the good scalability and can well support the group communication in the WAN environment.

2.6 Adam: A Library of Agreement Components for Reliable Distributed Programming [8]

This paper presents ADAM, a component-based library of agreement abstractions, used to build reliable programming toolkits. ADAM is based on a generic agreement component which can be configured to build many abstractions such as *Atomic Broadcast*, *Membership Management*, *View Synchrony*, *Non-Blocking Atomic Commitment*, etc. Currently, ADAM is used by the Eden framework, which is a group-based toolkit and also by OpenEden, which is an implementation of the Fault-Tolerant CORBA specification.

Questions and Answers

Is there any relation with other group communication system? We put the emphasis on a component approach and try to implement various agreement problem by configuring parameters.

How F, GET are provided? A concrete agreement component registers and offers the appropriate functions.

2.7 Anonymous Communication in Practice [3]

Anonymity is something most people expect in their daily lives. We usually expect to be anonymous when we vote, and, for example it is essential for many telephone hot-lines for people with serious problems that the users are anonymous. In general there are many daily scenarios where people would like to

communicate with others without letting any third party know who they are communicating with. Communication on the Internet is not anonymous, but if we want our activity online to be as private as offline, anonymity is needed. The goal of this paper is to provide a view on the field of anonymous communication with a focus on practical solutions to aid the designers of distributed systems where privacy is an issue. We explain the problem of anonymous communication and discuss so-called rerouting based solutions. Our discussion includes an overview of security issues and a survey of useful designs.

Questions and Answers

Is dummy traffic sometimes used to hide communication? Yes, dummy traffic is needed to prevent traffic analyzing. Tarzan uses pairs of rerouters that always sends the same amount of traffic. If no real traffic is available, random data is sent.

Does mixing scale? The short answer for most designs are yes. If the rerouters are organized in a decentralized way, it would not be a problem with lots of servers. The number of rerouters must however grow with the usage otherwise the system will obviously slow down, because the work load will increase.

How could the rerouters be organized? In theory they could be placed in one room, or we could even make a system with just one rerouter—in practice that would be to vulnerable. It would be good to spread out rerouters geographically as well as on different "groups" (nations, companies, organizations) to make the rerouting hard to attack by compromising rerouters.

What if compromised rerouters changes packet? One attack could be just to drop packets, this cannot be prevented. Another attack is to change the content before resending the packet but otherwise each packet should be integrity protected. This is actually easy to do, because often there is a layer of encryption for each rerouter, a checksum could be included before encryption, this could be verified after decryption.

How does "receiver" anonymity work? Usually it is done by the receiver providing information on a path in the rerouting network, the sender must obtain this before a communication could be initiated.

This needs some kind of Publish-Subscriber system!? Yes.

2.8 Communication Abstractions Through New Language Concepts

In this paper [5] we take the position that dedicated language concepts are to be considered as the solution for introducing commonly used communication abstractions into distributed programs. In our research we explicitly abandon middleware solutions, such as generation of stubs and skeletons. They do not give rise to the new ways of thinking that will be required for the construction of distributed and mobile systems in highly dynamic environments such as interconnected desktops, pda's and domotics. More specifically, we think that

both the complexity and weakness of most middleware technology and the 'solutions' the spawn is due to the fact that the technology is statically typed and class-based. Indeed, the major raison d'etre for generated interfaces and stubs is to satisfy type checkers for static languages. Because of that, we are starting to investigate how we can put the properties of prototype-based languages to structure and simplify the development of mobile agent software and thus also distributed systems.

In this paper we introduce some preliminary resulting communication abstractions based on the delegation mechanism most prototype-based languages feature. As a concrete case we will discuss some coordination problems in the master-slave design pattern and do a few *gedankenexperiments* in language design in this context.

Questions and Answers

Can class and prototype based languages be mixed? The prototype based language we are designing also mixes both concepts to avoid some of the problems associated with prototype based languages (cfr. Paper on Intersecting classes and prototypes on http://www.dedecker.net/jessie/publications), such as the usage of mixins to solve the problem of encapsulation with object inheritance. However, we do not have looked for a static type system while maintaining sufficient dynamicity of the program.

What are the semantics when the WE-construct is used in a child? When the WE construct is used in a child, then the message is sent to the children of that child. We are also exploring the possibility of adding another keyword that is the inverse of the super, that is to send messages to the direct children of a parent as opposed to sending the message to all children in the tree. Probably more clearer names should be chosen for these keywords.

Why introduce new language concepts and not implement them through the data abstractions that are already present in the language? There are several advantages of having dedicated language concepts that support the expression of communication abstractions: First, it can impose a new way of thinking to the developer. Second, it can influence the design of the program so it can aid in a better design of the program leading to a better reusability and understandability of the code. Third, compilation techniques can be used to optimize the use of these constructs depending on the context where they are used. However, we need to be very careful in our choice of what constructs we choose to add to our language. The constructs that are added to the language should be as orthogonal as possible with clearly defined semantics.

3 Discussions

The second part of the workshop was dedicated to discussions and working groups. After a brainstorming session where attendees suggested several subjects of discussion, we selected three of them for further exploration: communication abstraction taxonomies, quality of service, and peer-to-peer architectures.

3.1 Communication Abstraction Taxonomy

The "taxonomy of communication abstractions" working group was an attempt to (in an hour and a half) to organize the space of communication abstractions. The papers in the communications abstractions workshops cover a wide range of topics. This working group was an attempt to bring some regularity and organization to the communication abstraction space. Identification of the dimensions of the communications space has the additional advantage that many possibilities for communication organizations become evident by their place in the space. If, in some sense, a communication abstraction has dimensions such as "synchronicity," "object identity," and so forth, and the possible choices for each dimension can be identified, then the space itself defines a wide variety of models to be examined and characterized.

Communication abstractions are about modelling concurrent processing systems. Models serve as a foundation for analysis. By examining a model of communication, one can prove properties such as reliability, security, or nontermination. By analyzing the behavior of a model, one can derive metrics such as the temporal, space, or message efficiency of algorithms.

One thing that the discussion quickly revealed was that communication abstractions vary in their degree of detail and depth. By considering or defining additional features about a model, additional facts about that model can be derived. However, superfluous detail gets in the way of analysis of more fundamental properties. To reflect these distinctions, the group defined a "chemical hierarchy" of communication abstractions: the most basic models were the "quarks" of communication, assembled and extended successively into "atomic," "(simple) inorganic" modules, and most complexly, "organic" molecules.

Quark dimensions include:

Entities. This speaks to the kinds of things communicate. Most familiar object identity is communicants as discrete objects with identity.

Population dynamics. This dimension is the lifecycle of communicating entities. Examples of lifecycle issues include whether entities can be created or destroyed in the model.

The "atomic" dimensions speak to the primitive communication model. Subelements of this topic include:

Syncronicity. Whether the model supports synchronous or asynchronous communications (blocking and unblocking)

Sharing. Whether communication is by message passing or shared memory

Casting. Whether communication is monocast (directed at a particular recipient), multicast (directed at a defined set of recipients) or broadcast (receivable by everyone)

Signatures. Whether communications communications identify the sender.

Moving up the complexity chain, we can build communication models with the complexity of simple molecules. Molecular communication models add additional elements with respect to dimensions such as:

Failure. That is, what guarantees does the system make about communication delivery and what notification the system provides for communication failures. Failure notions also include consistency models for shared memory.

Arbitration. Who sees things in which order. A communication model might hypothesize, for example, that messages are received in the order sent or merely that a sent message will be eventually received.

Fairness. Does the model make any assertion about order of processing.

Structure of messages. What structure (for example, required or allowed fields, varieties of message types) does the model demand for message content

Direction of flow of information. In the communication act, does information flow from the initiator of communication to a target, from the target to the initiator, in both directions, or does no more information than synchronization get exchanged?

Streaming. Are messages discrete entities, or does the model support some form of "streamed" communications.

Coordination. What mechanisms does the model hypothesize for synchronization, coordination, or serialization among entities. For example, a model may hypothesize recognition of the "termination" of a set of "objects."

Model builders can develop models with organic complexity by embellishing models with additional features. These include:

Locus. A model may support a notion of physical locality. This may be a simple as distinguishing between elements that are "local" versus ones that are "remote," or might extend to an entire proximity structure. At the use level, this distinction may be seen in "access transparency" and "location transparency." Having introduced a notion of locus, we can also address mobility: the ability of an object to change its locus.

Connectors. A model might divide its entities into "communicators" and "connectors," or even make other distinctions in the entity space, with different actions and privileges ascribed to different kinds of entities.

Annotation. A model might provide additional meta-information about the communicating entities. Examples of such information include the protocols (interfaces) the entity supports, or its "ownership."

Conversations. A model may provide "conversational" mechanisms that support a pattern of communications. Examples of such conversational mechanisms include transactions and protocols.

Quality of service. A model may provide mechanisms for varying the quality of service to different entities.

Security. Having defined communication and message structures, a model may address properties such as guarantees of information reaching its destination, data integrity, anonymicity, eavesdropping, and identity spoofing.

Time. All the of the above issues may be extended with temporal notions, including, for example, the ability to put temporal limits or exceptions on other behaviors.

We note that the above discussion is shadowed by a kind of "Turing equivalence theorem." The distinctions are in some sense arbitrary, because choices in many dimensions may be used (in a sufficiently rich computational environment) to mimic the behavior of the opposite choices. For example, shared memory can be understood as a particular variety of messages to a memory entity; asynchronous communication can be mimicked using synchronous communication and secretary threads.

3.2 Quality of Service

One of the topics discussed during the workshop was the relationship between quality of service requirements, the separation of concerns, and communication abstractions. Obviously, engineers will benefit from abstractions that can be refined and whose quality of service can be customized to the needs of the application or the deployment environment. The central issues are where and when the necessary QoS parameters are determined, and how their provision is reflected in the communication abstractions.

A separation of concerns is reasonable in several respects. We should assume multiple layers of abstraction due to their interrelated but different focus: an application, a middleware, and a protocol layer. This facilitates simple, not too general communication abstractions that are focused on a specific layer. Different techniques may be applied on the abstractions to handle QoS requirements. For instance, aspect-oriented programming (AOP) and filter composition are readily available for application programming, while their applicability and relationship to reflective middleware and interceptors in the middleware layer is an open issue. On protocol and operating system layers even other approaches are expedient, cf. the x-Kernel.

A clean separation of QoS management from communication abstractions seems to have advantages, but cannot be achieved always and may even be undesirable. While leaving abstractions as simple as possible is reasonable, one may want QoS handling to be part of abstraction primitives/APIs for two reasons: to specify application requirements and for awareness of QoS degradation. The latter is important to enable an application to react to scarce resources, which mainly determines whether QoS should be integrated into the abstractions. Is the component itself not affected by QoS degradation or has it to know of the available QoS? At design time, compile/deployment time, or at runtime? Thus, depending on the availability of the afore-mentioned techniques to implement QoS handling in the different layers a number of different approaches for QoS-aware communication abstractions are conceivable, ranging from simple primitives that are implemented in different ways (e.g., send() methods that use either simple TCP or atomic broadcast services), to QoS aspects woven into application code wrapping the actual abstraction primitives, to QoS specifications that are traded and matched to existing services.

3.3 Peer-to-Peer

This discussion group may be seen as a continuation of the one of the last year, which was concluded as follows:

Peer-to-Peer (P2P) systems promote technological and social changes like new devices, new way of working, going both ways. P2P as a new communication patterns demand new solutions such as fault tolerant TCP/IP and no central authority. This kind of communication, which may ne anonymous or pseudonymous, leads to awareness, heterogeneous devices and more freedom ("cutting out the middle man").

Right now there is no consensus on the definition of P2P communication. All proposed definitions reflect its use rather than its concepts. Then our group focused its discussion on the concepts behind P2P communication. Here is the list of the concepts which we identified:

- no centralized control: This often leads to a greater system robustness.
- A symmetrical communication, which contrasts with the Client/Server model's asymmetrical communication.
- Nodes are servers and clients at the same time:
  - A peer takes the server role when it receives a client request,
  - A peer takes client role on external events, because the client initiates communication to servers,
- Peers are equal partners at the same level. There is no hierarchy between peers.

Another point was discussed by the group, concerns which abstractions are needed for P2P communication. Abstraction may relate to the following aspects:

- Mapping: how to map the peer ID to its location and how are IDs assigned.
- Peer discovery: how to discover other peers and negotiate communication channels.
- Asynchronous and event-driven communication: Several communications models may be taken as example. Among them we can quote broadcast (one to all), multicast (one-to-many) and publish/subscribe communication model.

These points can be taken as a starting point for future research.

4 Workshop Conclusions

Understanding and categorizing interactions among software entities appears to be a crucial in the future. This workshop attempts to gather people from various communities that are working on the same problem in order to meet and share their points of views. Communication is essential to multi-agents systems, software architecture, and distributed systems, but these fields have different goals.

This year, the workshop was more oriented towards platforms and implementation and adaptation of communication means.

Finally, we intend to setup a mailing list and a collaborative WikiWiki site in order to share ideas and projects[3].

The attendees suggested organizing a follow up workshop next year.

References

1. João Paulo Almeida, Marten van Sinderen, Luís Ferreira Pires, and Dick Quartel. A systematic approach to platform-independent design based on the service concept. to appear Sept. 2003.
2. J.-P. Almeida, M. van Sinderen, D. Quartel, and L. Ferreira Pires. Interaction systems design and the protocol- and middleware-centred paradigms in distributed application development. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.
3. C. Boesgaard. Anonymous communication in practice. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.
4. O. Ciuhandu and J. Murphy. A modular qos-enabled load management framework for component-based middleware. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.
5. J. Dedecker and W. De Meuter. Communication abstractions through new language concepts. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.
6. J. Kangasharju E. Aitenbichler. Communication abstractions in mundocore. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.
7. A. Frei, A. Popovici, and G. Alonso. Event based systems as adaptative middleware platforms. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.
8. F. Greve, M. Hurfin, J.-P. Le Narzul, Xiaoyung Ma, and F. Tronel. Adam: A library of agreement component for reliable distributed programming. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.
9. Yang Jia and Jizhou Sun. Cscwgroup: A group communication system for cooperative work in the wan. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.
10. D.A.C. Quartel, L. Ferreira Pires, M. van Sinderen, H.M. Franken, and C.A. Vissers. On the role of basic design concepts in behaviour structuring. *Computer Networks and ISDN Systems*, 29:413–436, 1997.

[3] If you want more information please contact antoine.beugnard@enst-bretagne.fr

.NET: The Programmer's Perspective

Hans-Jürgen Hoffmann

Department of Computer Science,
Darmstadt University of Technology, Germany
HJHoffmann@ACM.org

Abstract. Report about the ECOOP 2003 workshop WS 03, ".NET: The Programmer's Perspective". — Much of the push behind Microsoft's new .NET technology has been directed at such end-user applications as Web Services, but .NET also provides, through the ".NET framework", a set of tools and facilities of interest to software developers. This workshop had been set up to review pros and cons of the .NET framework as seen by programmers. Seven topics were discussed in much depth based upon preselected contributions of participants. More topics were shortly touched.

1 Outline of the Workshop

The workshop took place on Tuesday, July 22, 2003. It was a full day workshop. Following an introduction by H.-J. Hoffmann, Ms. Karine Arnout from ETHZ (taking the place of Prof. Bertrand Meyer, co-organisator of the workshop who was prevented from attending the workshop) started the presentations by an introductory talk *"Introduction to .NET and Eiffel for .NET"*; she introduced the salient features of the .NET framework and, besides covering C#, described - as an example of the multi-language support offered - the approach taken in the "Eiffel for .NET" implementation with emphasis on "Design by contract" as a well-known, important software engineering principle.

Harald Haller reported next on experience with and best practices gained in two .NET applications; the projects had a size of 10 – 20 person years and are considered to be rather successful.

Not mentioning all the details. the next two presentations by Diego Colombo, *"CIL + Metadata > Executable Program"*, and Bart Jacobs, *"Selection of run-time services in .NET: There is room for improvements"* allowed to learn about possibilities offered by the .NET metadata concept going beyond present solutions.

In the other three presentations by Anis Charfi, *"Software interactions"*, by Riccardo Casero and Mirko Cesarini, *"Managing code dependencies in C#"*, and Peter Tröger, *"Component programming with .NET"*, interesting and challenging perspectives for programming in the .NET framework were developed, again not mentioning all details.

All presentations led to many thoughtful discussions bringing to the participants a deep understanding of the pros and cons of the .NET approach. It remains to mention that the written contributions/presentations (see below in section 3, *Documentation*) list co-authors.

Following the presentations and their thorough discussion a general discussion period of about 1 ½ hours was scheduled. Section 2 lists the main topics discussed.

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 30–33, 2004.

2 Topics Discussed

During the individual discussion of the presentations a number of relevant topics came up. Arguments *Pros/cons* may be found already in the documented contributions (see WWW site mentioned below) insofar as authors had addressed them beforehand. Some of the topics were rather specific in the scope of the presentations considered. In the general discussion period at the end of the workshop they were altogether included if considered to be of a broader interest.

In addition a "to-discuss-list" had been prepared before the workshop.

- .NET middleware architecture, experience report
- .NET and Corba / J2EE / Com+
- Visual Studio.NET as a programming environment
- .NET and Component composition
- .NET and deployment
- Meta-data
- C# and Aspect-oriented programming
- C# and Design by contract / Programming by contract
- Shift C++ / Java ==> (.NET /) C#
- .NET and Open Source ?
- Internationalisation

We categorised the result of opinions brought up into *positive*, *neutral* and *negative*. The picture given should not be understood as truly objective; it represents individual experience, expectation, and background. The entries in the lists highlight the outcome of a time-limited ad-hoc discussion. Thus the lists should not be considered to be complete and fully supported by hard objective arguments.

Positive:

- Good incorporation of concepts of object-oriented programming
- Common Type System
- Metadata and Attribute treatment
- Standard architecture over different platforms
- Language interoperability
- Smooth transition between value/reference
- Component composition facilities
- Deployment support

Neutral:

- Web services
- Dependency processing
- Provisions for dynamisation of program behaviour during execution
- Aspect-oriented programming possible
- VisualStudio team environment became acceptable

Negative:

- Lack of native support for engineering environments
- Design specification following *Design by Contract*-technology not supported
- Customisation of client devices not possible
- *Open Source* co-operation in programming needs more support
- Shifting from C++/Java to C# not yet totally solved

3 Documentation

All accepted contributions of participants are available under the URL
http://www.informatik.tu-darmstadt.de/PU/ECOOP/Submissions.
Some of the ppt-files presented during the workshop are there available as well. In addition there is a collection of .NET-related abbreviations often found in the literature
http://www.informatik.tu-darmstadt.de/PU/ECOOP/NEWS/Abbreviations.PDF.
It is planned to have a forthcoming issue of the electronic Journal of Object Technology (JOT, http://www.jot.fm/issues/...), published by the Swiss Federal Institute of Technology, publisher Prof. Bertrand Meyer, devoted to the workshop.

4 Participants

Regular participants of the workshop (in alphabetic order) were:

Karine Arnout (representing co-organisator Bertrand Meyer), ETHZ, Switzerland
Riccardo Casero, Politecnico di Milano / DEI, Italy
Mirko Cesarini, Politecnico di Milano /DEI, Italy
Anis Charfi, Univ. Nice / CNRS, France
Diego Colombo, Univ. of Pisa, Italy
Harald Haller, sd&m, Munich, Germany
Hans-Jürgen Hoffmann – organisator –, Darmstadt Univ. of Technology, Germany
Bart Jacobs, K. U. Leuven, Belgium
Mattia Monga, Univ. degli Studi di Milano, Italy
Michel Riveill, Univ. of Nice, France
Doris Schmedding, Univ.of Dortmund, Germany
Wolfgang Schult, HPI, Univ. of Potsdam, Germany
Peter Tröger, Hasso-Plattner-Institute, Univ. of Potsdam, Germany

Further persons attended the workshop as guests without being introduced beforehand by a submitted and reviewed paper (some of these persons attended only partially – misspelled names from hand-written list excepted ! –):

Davide Ancona, DISI, Univ. of Genova, Italy
Andrew Cain, Swinborne; Australia
Sari Eldadah, Arab Academy for Banking & Financial Science, Jordan
Mei Feng, Newcastle Univ., UK
Carl Gunter, Univ. of Pennsylvania, USA
Uwe Hohenstein, Siemens AG, Munich, Germany
Kim Jin-Young, Ojou Univ., South Korea
Boris Litvac, Tel-Aviv Univ., Israel
Giovanni Lagovio, DISI, Univ. of Genova, Italy
Kim Myung-Uk, Ojou Univ, South Korea
Sorin Moldovan, Univ. of Babes-Bolyai, Romania
David Prasad, MUNIT, India

Pawel Stowikowski, AGH, Univ. of Cracow, Poland
Bart Verheecke, Vrije Univ. Brussel, Belgium
Gansha Wu, Intel Chinese Research Center, China
Victor Ying, Intel Chinese Research Center, China

5 Conclusions

A fair discussion of many of the challenging advancements of .NET took place during the workshop. For the regular participants with planned/scheduled contributions a forum for presentation of their ideas and results of research was provided, allowing competent discussion by the whole group. Some of the guests, especially in the afternoon sessions and the general discussion period, brought interesting and helpful arguments into the discussions.

Eiffel .NET, introduced in the beginning, was in many cases something like a benchmark demonstrating how the .NET advancements may be measured from a "pure" object-oriented (language) point of view. .NET as a middle-ware system proved as an interesting achievements in component technology. .NET overall, although not free of an inherent level of complexity, was considered as a remarkable step of improvement in software/system technology; even if in details some critics came up.

We had very strong and competent contributions from industry; however, more participation by industry researchers would have been estimated. Contributions by persons from academic institutions identified interesting positive aspects and exhibited missing links to available and important scientific knowhow with respect to the state-of-the-art in software engineering. Thanks to all of you!

Special thanks go to the members of the reviewing committee:

- Robert L. Baber University of Limerick, Limerick, Ireland
- Johannes Heigert Munich Univ. of Applied Sciences, Munich, Germany
- Erhard Ploedereder University of Stuttgart, Stuttgart, Germany
- Michel Riveill Université de Nice, France
- Christian Salzmann Technische Universität München, Munich, Germany
- Doris Schmedding University of Dortmund, Dortmund, Germany
- Jürgen F.H. Winkler Friedrich Schiller University Jena, Jena, Germany

We acknowlege support by the executive committee of the German Chapter of the ACM. Special thanks to the members of the editorial board of JOT and Prof. Bertrand Meyer as publisher for providing a possibility of publication of the results.

Component-Oriented Programming

Jan Bosch[1], Clemens Szyperski[2], and Wolfgang Weck[3]

[1] University of Groningen, Netherlands
Jan.Bosch@cs.rug.nl – url http://www.cs.rug.nl/~bosch/
[2] Microsoft, USA
CSzypers@microsoft.com – urlhttp://research.microsoft.com/~cszypers/
[3] Oberon microsystems, Switzerland
Weck@oberon.ch

Abstract. This report covers the eighth Workshop on Component-Oriented Programming (WCOP). WCOP has been affiliated with ECOOP since its inception in 1996. The report summarizes the contributions made by authors of accepted position papers as well as those made by all attendees of the workshop sessions.

1 Introduction

WCOP 2003, held in conjunction with ECOOP 2003 in Darmstadt, Germany, was the eighth workshop in the successful series of workshops on component-oriented programming. The previous workshops were held in conjunction with earlier ECOOP conferences in Linz, Austria; Jyväskylä, Finland; Brussels, Belgium; Lisbon, Portugal; Sophia Antipolis, France; Budapest, Hungary; and Málaga, Spain.

WCOP96 had focused on the principal idea of software components and worked towards definitions of terms. In particular, a high-level definition of what a software component is was formulated. WCOP97 concentrated on compositional aspects, architecture and gluing, substitutability, interface evolution, and non-functional requirements. WCOP98 had a closer look at issues arising in industrial practice and developed a major focus on the issues of adaptation. WCOP'99 moved on to address issues of structured software architecture and component frameworks, especially in the context of large systems. WCOP 2000 focused on component composition, validation and refinement and the use of component technology in the software industry. WCOP 2001 addressed issues associated with containers, dynamic reconfiguration, conformance and quality attributes. WCOP 2002 has an explicit focus on dynamic reconfiguration of component systems, that is, the overlap between COP and dynamic architectures.

WCOP 2003 had been announced as follows:

> COP has been described as the natural extension of object-oriented programming to the realm of independently extensible systems. Several important approaches have emerged over the recent years, including

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 34–49, 2004.

CORBA/CCM, COM/COM+, J2EE/EJB, and most recently .NET. After WCOP'96 focused on the fundamental terminology of COP, the subsequent workshops expanded into the many related facets of component software.

WCOP 2003 will emphasize the dynamic composition of component-based systems and component-oriented agile development processes. A typical example of dynamic, i.e. run-time, composition of systems is the notion of web-services, but also in other domains and contexts this trend can be identified. This requires clearly specified and documented contracts, standardized architectures, specifications of functional properties and other quality attributes, and mechanisms for dynamic discovery and binding of services. A service is a running instance (all the way down to supporting hardware and infrastructure) that has specific quality attributes, such as availability, while a component needs to be deployed, installed, loaded, instantiated, composed, and initialized before it can be put to actual use. The commonalities and differences between service and component composition models are interesting and a proposed focus of this workshop.

Agile development processes can benefit from component-based development in that use of existing or of-the-shelf components reduces the amount of required development effort and gives quick results early in the process. This requires deciding early in the process which specific architectures, standards, interfaces, frameworks or even components to use. Unfortunately, such early decisions contradict general agility, as promoted, for instance, by extreme programming, because reconsidering such fundamental decisions later in the development process comes at considerable cost but may be unavoidable at the same time. On the one hand, one may end up with developing a new component instead of deploying an of-the-shelf component as planned earlier. On the other hand, a chosen architecture or interface may proof inadequate later in the process, when additional requirements are considered for the first time. As a source of delay and extra cost this easily puts the entire development at risk. Can these contradictions be dealt with?

COP aims at producing software components for a component market and for late composition. Composers are third parties, possibly the end users, who are not able or willing to change components. This requires standards to allow independently created components to interoperate, and specifications that put the composer into the position to decide what can be composed under which conditions. On these grounds, WCOP'96 led to the following definition:

> A component is a unit of composition with contractually specified interfaces and explicit context dependencies only. Components can be deployed independently and are subject to composition by third parties.

Often discussed in the context of COP are quality attributes (a.k.a. system qualities). A key problem that results from the dual nature of components be-

tween technology and markets are the non-technical aspects of components, including marketing, distribution, selection, licensing, and so on. While it is already hard to establish functional properties under free composition of components, non-functional and non-technical aspects tend to emerge from composition and are thus even harder to control. In the context of specific architectures, it remains an open question what can be said about the quality attributes of systems composed according to the architecture's constraints.

As in previous years, we could identify a trend away from the specifics of individual components and towards the issues associated with composition and integration of components in systems. While the call asked for position papers on the relationship to agile development processes, we did not receive any such papers. An emphasis on anchoring methods in specifications and architecture was noticeable, going beyond the focus on run-time mechanisms in the previous year. Sixteen papers by authors in nine countries were accepted for presentation at the workshop and publication in the workshop proceedings; three submissions were rejected. About 35 participants from around the world participated in the workshop. The workshop was organized into four morning sessions with presentations, one afternoon breakout session with four focus groups, and one final afternoon session gathering reports from the breakout session and discussing future direction.

2 Presentations

This section summarizes briefly the contributions of the sixteen presenters, as grouped into four sessions, i.e. Specification and Predictable Assembly, MDA and Adaptation, Separation of Concerns, and, finally, Dynamic Reconfiguration.

2.1 Specification and Predictable Assembly

The first paper, presented by Sven Overhage, is a proposal towards a standardized specification framework covering five specification perspectives, following the UDDI example of grouping these into 'colored' specification pages. In particular, the following five colors are proposed: (i) general and commercial information (White Pages), (ii) component classifications (Yellow Pages), (iii) component functionality (Blue Pages), (iv) component interfaces (Green Pages), and (v) implementation-related information (Grey Pages). Each of the last three perspectives is addressed from the static, operations, and dynamic views. Blue pages cover a lightweight ontology over concepts such as entities, tasks, processes, etc. Green pages specify provided and required interfaces, assertions, and use temporal logic for protocols. Grey pages use QML (quality modeling language) to specify extra-functional properties. As future directions, the paper proposes (i) a meta-model to integrate different notations, (ii) standardization, (iii) the implementation of specification tools and repositories, and (iv) the development of a multi-dimensional component type system for compatibility tests.

The second paper on contracts and quality specifications of software components was presented by Ralf Reussner. Starting from a situation of wide-spread interest in quality-of-service (QoS) and work on QoS specification languages (such as QML), the paper identifies two problems: (i) a component has to interrogate the environment for QoS in order to offer QoS (thus a need for requires-interfaces for QoS) and (ii) a component vendor has to select which QoS properties to specify in provides and requires interfaces. (Reussner noted that a requires interface can be seen as a precondition and a provides interface as a postcondition to the component assembly function.) QoS is context dependent: (i) there is no single value for QAs of a component; (ii) a component exposes different quality in different contexts; and (iii) the component vendor cannot specify a QA as a constant. These observations lead to the concept of deployment-time interfaces for components that reflect context properties. Deployment-time evaluation of a parameterised contract can be used to perform static interoperability checks. The authors found that many such parameterised contracts can be generated by code analysis (and guessing transition propabilities in resulting Markov models), still yielding reasonably accurate QoS predictions.

The third paper, presented by Franz Puntigam, discusses the use of state information in statically checked interfaces. The idea is to equip types with state variables and methods with pre- and poststates. The paper includes a series of small examples, including concurrency scenarios such as dining philosophers. The system supports static type checking and separate compilation. Several simultaneous references are supported. Alternative When-clauses can be used to state complex conditions. Dependent state information and exclusive use can be handled. In application to the area of component-based systems, the approach can be used to partially capture non-trivial protocols. A specific application discussed (but not fully resolved) in the paper is to determine safe points and mechanisms for type-state recovery for component hot-swapping. The last paper of this session, presented by Nicolae Dumitrascu, presents a methodology for predicting the performance of component-based applications. As a primary motivator, the authors identify the cost benefits of detecting problems early in a development effort. Their concrete examples are set in the context of .NET. Performance profiles of the used components are used to predict the performance of a component assembly (not of a .NET assembly); a difficulty is to properly account for run-time behavioral interaction. The presented approach predicts performance of an application taking into account: application design, resource design, assembly performance profile, and connection performance profiles. Extended queuing models are used as performance models.

2.2 MDA and Adaptation

In the first presentation of this session, Seung-Yun Lee presented their paper on an MDA-based COP approach. In their previous work, the authors had worked on CBD support: (i) generating EJBs from specifications, extracting EJBs from Java applications, extracting EJBs from databases; (ii) adapting and composing

EJBs; (iii) assembling components based on architecture diagrams; (iv) deploying assemblies; and (v) testing interfaces, running applications, and measuring performance. However, they identified the problem that CBD does not directly support interoperation and integration across different platforms. This leads to the viewpoint of model-driven architecture (MDA) as an evolutionary refinement of CBD. MDA works over the core features of a model repository, a set of model transformations, and a meta editor. Further features are required to support CBD: component identification from business modeling results and an architecture-centric approach. To enable this integration, the presented research project aims to deliver a full stack of MDA-based CBD tools.

In the second presentation Olivier Nano looked into ways to using MDA to integrate services in component platforms. Most current component platforms offer services such as persistency, transactions, or authentication. However, components are frequently required to target used services explicitly, which requires knowledge as to where to call a service and what arguments to pass. The claim is that the "join points" where service calls need to be inserted are independent of any component platform. To introduce systematic join points, a detailed meta-model of the invocation process is used with four interception points on the forward path (send, accept, receive, execute) and two on the return path (sendBack, return). Service integration is then described as interceptions along these points defined by the behavioral meta-model. Merging is performed by proved merging rules that are commutative and associative. The mapping to a concrete component platform is not a problem where such a mapping is surjective. In a next step, Nano proposed to try the construction of the model as a UML2 profile and to implement the transformation with QVT.

The third presentation was delivered by Abdelmajid Ketfi, focusing on dynamic interface adaptability (http://www-adele.imag.fr/ ketfi). The approach is to introduce dynamic adaptation frameworks for EJB and OSGi. Strategies and Monitoring (of material resources) form inputs to an adaptation manager, which modifies non-functional services that interact with a component's functional services; both functional and non-functional services provide feedback to the adaptation manager. An abstract application model is used (i) to extract meta-level information from an application and (ii) to describe an application at the meta-level. A prototype for Java has been developed. Dynamic interface adaptability poses a number of issues, including The need to handle syntactic dependencies and dynamic availability. Ketfi proposes to reify connections, that is, to introduce connectors as first-class entities ("aware connectors"). As future work, Ketfi named the dynamic adaptation of extra-functional services. After the presentation, he answered the question as to how dynamic Adaptation can be in his system by stating that he can adapt in a running system in principle, but that this depends on the concrete component platform used.

The fourth and final presentation of this session, presented by Gustavo Bobeff, looked into the "molding" of components using program-specialization techniques. The context of this approach is component development (where the stages of production, assembly/configuration, and execution are distinguished). Bobeff

observes that today adaptation is limited to the superficial level of adapting interfaces to the context of use, where implementations remain unchanged. His goal is to adapt the implementation to the context of use, where the idea is to rely on program transformation, using one of two particular techniques: partial evaluation or slicing. To be successful, Bobeff claims that the trick is to capture specialization opportunities at production time, moving deliverables from components to component generators that carry self-contained specialization knowledge. The component-specialization process then works by invoking component generator with context arguments at deployment time. The result is that deployed components have an adapted interface as well as implementation ("deep adaptation"). The question from the audience as to how specializing components would differ from specializing programs was addressed by noting that the individual suppliers of components do not communicate, but that the specialization should yet support the integration scenarios where multiple such components meet. (The created knowledge gap is closed by specializing at deployment and not at production time.)

2.3 Separation of Concerns

The first presentation of this session was by Paolo Falcarin. It dealt with technology supporting dynamic architectural changes for distributed services, in particular replacement of the middleware and dynamic evolution. For this, source code should be checked against an architectural description. In JADDA architecture is described in an XML-based language. At runtime, applications register with a system administrator component and receive such an architecture description in return. Connections to components are made using a name server. To replace the middleware, the administrator will distribute new architecture files and bindings are reestablished calling the name server again. When applications reconnect, is under their individual control and they can make sure to be in a consistent state at the time.

The second presentation, given by David Lorenz, discussed how technology of aspect-oriented programming can be used to em unweave legacy systems. Components can be em unplugged by using an em around advice that redirects calls to certain methods and handles them as an aspect rather than a component call. The still challenging part of this work is aspect interaction – as it is with the usual forward AOP.

Third, William Thomas talked about the need to improve verification techniques so that they extend to creating systems from existing components. Some policies must be enforced to create verifiable systems, but these policies often will depend on the application domain. Considerations are in particular when to activate mediators, the level of access to the application's state, and actions to be performed by the mediator. Also, mechanisms for detection and resolutions of policy conflicts need to be investigated.

Finally, Marek Prochazka presented a paper about the interaction of components and transactions. Transactions are a key service to components but each component framework and application server handles things differently. To

open the middle ground between component-controlled and container-controlled transactions, the proposed Jironde framework introduces separate controllers that can be associated with components. Controllers decide about transactional functionality.

2.4 Dynamic Reconfiguration

Dynamic Reconfiguration The fourth and final presentation session was concerned with the *dynamic reconfiguration* of component-based systems. The challenge of dynamic reconfiguration is to balance the freedom allowed after the deployment of the system versus the functionality and properties of the system that should remain available in the face of reconfigurations.

As the other sessions, also this session consisted of four papers. The first paper was presented by Kris Gybels and presents an approach to enforce feature set correctness. The paper starts out from a generative programming approach and extends this towards dynamic generative programming. It views a software system as consisting of a set of features and a set of composition rules. During reconfiguration, the existing set of composition rules should not be violated. The author identifies that, due to the interaction between the base language and the meta language describing the composition rules, one requires a *linguistic symbiosis* approach. In the presentation, the author presented a new version of the SOUL language that achieves a level of linguistic symbiosis with Smalltalk.

The second paper was presented by Vania Marangozova and discusses the use of replication as a means to achieve high quality component-based systems, especially from an availability perspective. The motivation is that one, on the one hand, would like to reuse functionality of different applications and, on the other hand, would like to use these applications in different execution contexts. A reusable and effective approach to high availability needs, thus, to be separated from the application code. The approach proposed by the author uses a simple component model for business applications, an application independent protocol design model and a composition model that facilitates the composition of the business application and replication protocol. The protocol controls the application, but the configuration is generated at system deployment. The advantages of the approach are the aspect separation between business and replication code, a component model of replication management and non intrusive approach to replication configuration. The disadvantages are the negative impact on performance and the, currently unclear, interaction with other system services.

The third presentation, by Mircea Trofin, presents a self-optimizing container design for enterprise Java beans applications. The paper addresses the problem that applications that run on a server cluster need to be modularized in order to support their evolution. On the other hand, especially fine-grained modularisation has, in most component interconnection standard implementations, a substantial negative impact on performance. The approach proposed in the paper focusses on optimizing containers. It uses call path and deployment

descriptor information, predetermines the services that are required and predetermines the state management requirements. Based on this, the approach is able to determine and define safe optimizations. The current implementation shows promising results in that response times are reduced with 20% to 40%.

The final paper in the dynamic reconfiguration session was presented by Jasminka Matevska-Meyer. The paper presents an approach to determine runtime dependency graphs and using this information to speed up runtime reconfiguration of component-based systems. The problem addressed by the paper is that static dependency graphs are overly negative in most situations, limiting the set of possible reconfigurations. The approach to enabling reconfiguration of component-based systems at runtime consists of three main steps. First, the running system is analyzed for a particular time interval with respect to the estimated reconfiguration duration. Second, it creates change-request specific runtime dependency graphs, based on specified (or derived) architecture and component protocols. Finally, it observes the running system at a particular time interval for the actual reconfiguration duration and then performs the reconfiguration. The key benefit of the approach are that the set of components affected by the reconfiguration is smaller, resulting in a larger set of services to be available during reconfiguration. This allows one to minimize the down-time of the system at reconfiguration.

3 Break-Out Session

During the afternoon, the attendees were organized into break-out groups. The breakout groups addressed a number of topics, i.e. interface specification and MDA, connection of components, dynamic reconfiguration, and separation of concerns. At the end of the day, the workshop was concluded with a reporting session where the results of the break-out sessions were presented. Below, the results of the sessions are summarized briefly.

3.1 Interface Specification Formalisms and MDA

The first breakout group discussed issues around interface specification and model-driven architecture (MDA). Arne Berre served as the scribe for this group and contributed the material for this subsection.

The group structured the topics to be discussed into three areas: (i) component specification frameworks, (ii) the interface model, and (iii) domain-specific standards.

Component Specification Frameworks. The group considered it an advantage if component specifications could be based on (extended from) standardized specification approaches such as the standards from OMG: UML 2.0 with its new component model (with a conceptual basis in the CCM), UML Quality/Fault-tolerance profile standard for QoS specifications, and classification models, such as the UDDI white/yellow/green pages. Specifications should be goal driven,

meaning that there should be a clear intention by each part of a specification to support its intended usage.

Three main usages were identified:

Component retrieval (to identify components – or candidates, respectively - based on component classifications like the yellow pages from UDDI) – In so doing, there is a need for similarity metrics on components, which needs more research.

Component Interoperability (for checking if two components can work together) – There is also a need for defining the semantics/meaning of concepts and method names, (ref. later on blue pages).

Component adaptation, configuration and generation – There is also a need to have enough information to be able to create adapters, to reconfigure components and to generate platform specific models. A question is which information needs to be attached to the interface.

The Interface Model. An interface model for business components from a working group of the German Society of Computer Science (GI) was accepted by the group as a principal basis for structuring component specifications. It builds upon the UDDI specification approach for (web) services and uses green pages to specify interface definitions and protocols, white pages for general business information, and yellow pages for classification information. The two additional "color" dimensions are "blue" pages for domain/business information and "gray" pages for quality information.

Component composition and logical structure/architecture is also important to be able to integrate into the specification framework, but this is the focus of another Working Group, so it was not discussed further.

A graphical overview and further information can be found from the works of the WG on Business Information Systems/GI (http://www.fachkomponenten.de (also in English)).

The MDA approach (From OMG Model Driven Architecture) could build upon these by capturing the color aspects in UML, with appropriate stereotypes/UML profiles. Subtyping should be possible in each of these (colors) dimensions. More research into static checking of protocol subtypes is required. The new UML 2.0 is viewed as a good basis for this, but further standardization is needed here, such as a platform independent type system, links between the Component model and Activity model, and profiles for the various "color" dimensions. The before suggested Interface model for business components (Fachkomponenten), might not yet cover all the aspects and have the precision required from MDA-based model transformation and code generation.

Specifications must be usable from tools, through a representation in XML/XMI. Hopefully the new version of XMI (version 2) will improve on the compatibility aspects that has been a problem so far.

Component specifications are dependent on the deployment/run-time context, and thus requires parameterized contracts, and the handling of dependency between required and provided interfaces. A component will typically have mul-

tiple interfaces/roles (deployment time interfaces) for the various contracts it is responsible for/involved in.

Domain-Specific Standards. In the context of discussing specification formalisms there was also a discussion on the conditions for such formalisms to come into practical usage.

It is not possible to mix and match components arbitrarily, it is necessary to have a common foundation in the understanding of the semantics/meaning of the concepts involved in the component interfaces and protocols. This can only be done by creating domain specific models/ontologies of concepts that can be referred to in the interface specifications.

An example that was discussed is the General Ledger domain, which has been addressed by the GI working group and was suggested as an OMG component standard (though unsuccessful so far). In the specification of the semantics it is for instance important to state whether a balancing operation is done according to US GAAP or IAS accounting principles.

Emerging component markets not only depend on technical standards such as specification formalisms, but also on market conditions that encourage companies to be involved in the agreement on such domain models.

For the companies that currently have a major presence in the market it can be difficult to find the incitement to take part in the promotion of open standards/ open models in their domain. This is more obvious as an opportunity for second generation companies.

This also relates to the more general problem of encouraging reuse, even within companies, with the NIH (Not invented here) syndrome etc.

3.2 How to Connect Components?

The second breakout group discussed issues regarding the definition of the connection of components prior to their deployment. The group's scribe was Markus Lumpe, who contributed the material for this subsection. In addition, the following attendees participated in this group: Selma Maliugui, Jorgen Steengaard-Madsen, Jacques Noyé, Seung-Yun Lee, and Gustavo Bobeff. The main subjects discussed where: (a) soundness of component connection, (b) architectural and behavioral aspects of connector specification, And (c) language support.

Component connection means that one connects required with provided services of components. When using component connectors, it was argued that it is necessary to check the "soundness of connection", which guarantees that (a) all involved types match, (b) the behavioral characteristics match, and (c) any additional constraint (e.g. real-time demands) is satisfied. Soundness checks must be done at both compile-time and runtime. However, a complete soundness check might not be feasible. Therefore, the breakout group members opted for an approach that allows for "partially erroneous connector specification" – a scheme that requires additional runtime checks, but allows a relaxed specification of component connections.

In order to address architectural and behavioral aspects of component connections, the members of the breakout group proposed to represent connectors as components. This will allow for a homogenous view of a component-based system and will facilitate reasoning about composition. In fact, one can think of a component-based system as a triangle. At the top node a component-based system consists only of one component that actually represents the whole architecture of the corresponding application. A zoom operation can then be used to decompose the application, that is, architectural views will incrementally be replaced by behavioral views. At the bottom only behavioral aspects remain.

To facilitate the specification of component connections, the members of the breakout group finally agreed on needing special language support. More precisely, there is a need for both a component language and a composition language. While the former should provide abstractions to specify components, the latter has to provide abstractions to specify component connections (i.e., component composition). Ideally, one would like to have a unified language that incorporates all aspects of component-based software development.

In the succeeding discussion in the plenum, it was noted that a soundness check of component connection may not be computable. A solution to this problem could be an approach that uses a combination of configurable compile-time and runtime checks.

3.3 Dynamic Reconfiguration of Systems (Post-deployment)

The third breakout group covered dynamic reconfiguration of component systems in the post-deployment phase. The group scribe was Jean-Guy Schneider, who contributed the material for this subsection. In addition, the following attendees participated in this group: Robert Bialek, Paolo Falcarin, Abdelmadjid Kefti, and Jasminka Matevska-Meyer.

To facilitate discussion of issues, the group first defined some terms: (i) a component is a state-full first class entity that has a well-defined interface (consisting of both provided and required services) and needs to be instantiated before use and (ii) a configuration to describe both the connections between provided and required services as well as the interaction protocols used for intercomponent communication.

The following questions were addressed during the breakout session:

- What is dynamic reconfiguration? What kinds of reconfiguration exist?
- Why is dynamic reconfiguration needed? Are there any examples which clearly indicate the need for dynamic reconfiguration?
- When is dynamic reconfiguration performed?
- Who does dynamic reconfiguration?
- How is dynamic reconfiguration performed?

What is dynamic reconfiguration? In essence, the workout group agreed that (i) a change of connections and bindings between components and (ii) a change of the respective communication protocols are the two main reconfiguration mechanisms. The group also observed that a strict separation between application

logic and (re)-configuration logic is necessary. Taking this into consideration, "patching" a component (i.e., adding new or changing existing behavior) should not be considered as a reconfiguration, although it will probably have an effect on the behavior of the overall system.

Why is reconfiguration of a running system needed? Various reasons were brought forward: (i) a change of some third-party component may trigger a reconfiguration in the remaining system, (ii) a change in the deployment environment requires an adaptation of communication protocols (e.g., change in network protocols), (iii) improving the overall performance of a system requires a different configuration, (iv) adding fault tolerance to a system, and (v) monitoring a running application.

What are characteristic scenarios that require dynamic reconfiguration and cannot be implemented using "traditional" approaches? The following list was defined during the discussion: (i) service adaptation in distributed systems, (ii) mobile-phone networks (in particular exchanging one base station for another), (iii) embedded systems, (iv) fault-tolerant systems (changing servers due to system load etc.), and (v) evolution of web-applications. What about personalization of (web-)services? Taking the separation of configuration and application logic into account, it was agreed that this should not be considered as an example of reconfiguration, but more as a case of application "patching."

Who initiates dynamic reconfiguration? It can either be an external entity or the system itself. In the latter case, access to the system environment is needed (e.g., to find out about loads of servers).

When is a running system reconfigured? Possible time points for reconfiguration can be derived from the system specification or by monitoring the running system and determining a "save" time point based on its execution state.

During the breakout discussion, the participants noted issues which were not directly related to any of the questions, but had to be addressed in the context of dynamic reconfiguration. Most important is the fact that any reconfiguration of a system needs to leave the system in a consistent (configuration) state. One way of achieving this goal is to view reconfigurations as transactions that can be rolled-back if the system does not end up in a consistent state. In the succeeding discussion in the plenum, it was noted that there is no real need to have transaction support for dynamic reconfiguration as this is simply one approach to ensure that any reconfiguration will leave the system in a consistent state. However, it is very probable that some form of reification support is needed for consistency checks. Finally, it was suggested that it would be possible to use concepts from aspect-oriented programming to incorporate support for unexpected configuration change.

3.4 Separation of Concerns

The fourth break-out group addressed various issues around separation (and composition!) of concerns. The group's scribe was Vania Marangozova, who contributed the material for this subsection.

An alternative session title: "How six researchers working in different domains define the key issues of the problem." The groups participants were: Kris Gybels (Vrije Universiteit, Brussel, Belgium), working on languages; William Thomas (The MITRE Corporation, VA, USA), working on interception techniques in legacy systems for policies; Marek Prohazka (INRIA Rhône-Alpes, France), working on adding and configuring transactions; Olivier Nano (Université de Nice-Sophia-Antipolis, France), working on adding system services to an existing system; Mircea Trofin (Dublin City University, Ireland), working on container configuration; and Vania Marangozova (INRIA Rhône-Alpes, France), working on adding and configuring replication.

The discussions in this group were organized around three subjects. In the first place, the participants tried to identify the key issues related to separation of concerns. In the second place, they tried to propose a classification of the approaches that handle these issues. In the third and last place, participants have tried to conclude on the feasibility and the applicability of separation of concerns. These three areas are detailed further in the following.

Key Questions. To provide separation of concerns, the following three questions need to be addressed:

1. The What Question: What is a concern? Without a clear definition of what a concern is, it is very difficult to provide tools for separation of concerns!
2. The When Question: When are concerns separated? In particular, when does one think in terms of separated concerns? When are concerns composed? When is information about composed concerns available?
3. The How Question: How is separation of concerns achieved? What are the techniques?

What is a concern? There are two main approaches to defining the nature of concerns: the first one is adopted by the system-oriented community while the second approach is typical of language-oriented works. The system-oriented approach is interested in the modular management of system services such as transactions, security, fault tolerance, etc. A concern corresponds to a system service and the goal is to define the management of a system service separately from the business (non system) code, as well as separately from the management of other services. The language-oriented approach does not make any distinction between system and non-system concerns. It considers that every treatment may be considered as a concern. It follows the very generic definition given by Dijkstra. Given that, from a different point of view, the implementation of a system service can be considered as a business treatment, this general definition seems to be a better one.

When are concerns separated? To be able to mange separate concerns, it seems necessary to think in terms of such separate concerns already at design time. Separate concerns may be composed during different phases of the application's lifecycle. They can be composed statically, during design. They can also be composed during a configuration phase before the actual execution of

an application. Finally, there may be a need to change the composition of concerns during runtime. The moment of composition of concerns is related to the management of the (meta) information about these separate concerns. In fact, a system where composition is done once and is not subject to change does not need to keep the information of the concerns to be composed. On the contrary, if a system needs to undergo changes during execution, it will certainly need the information characterizing the separation of composed concerns.

How are concerns separated? As there are two definitions of what concerns are, there are also two approaches to manage separation of concerns. In the case of language-oriented works, concerns are expressed in terms of programming instructions and the separation techniques are related to code manipulation and transformation (AOP). As these techniques are working on a very detailed level, they are very powerful. However, the power of expression induces complexity. Another problematic point with this kind of techniques is that they suppose to be able to access and modify the source code. This is an issue since components are supposed to follow the black-box principle. The system-oriented approach has the opposite characteristics. It works on structural (architectural) entities and separation of concerns is more commonly obtained through interception at the boundaries of these entities. Compared to the language approach, there are fewer entities to consider, so it is simpler to manage. However, such structural models induce limitations on the set of concerns and separations that can be managed.

Feasibility and Applicability (Open Questions). In order to separate concerns, it is very important to identify hidden assumptions like assumptions on the execution system, the programming paradigm, etc. It seems very difficult to program a concern which is really generic and which does not have any dependencies on other concerns. Moreover, it seems natural for a programmer to specify constraints on other concerns! The group concluded that separation of concerns is not feasible in all cases. For example, in the cases of system services like transaction management and replication there is a need to modify the "other" code in a specific way in order to make the system work. Given this, to push the idea of separation of concerns further, there is a need to further consider the following two points:

- Identification of different types of concerns. We will need to define more precisely the concerns to be composed in a system in order to provide the adapted tools for their management.
- Identification of dependencies. For each type of concern, we will need to define the dependencies between this concern and other code. This will help understanding of concern's organization and the way they need to be composed with other code.

4 Final Words

As organizers, we can again look back on a highly successful workshop on component-oriented programming. We are especially pleased with the constantly evolving range of topics addressed in the workshops, the enthusiasm of the attendees, the quality of the contributions and the continuing large attendance of more than 30 and often as many as 40 persons.

We would like to thank all participants of and contributors to the eighth international workshop on component-oriented programming. In particular, we would like to thank the scribes of the break-out groups.

5 Accepted Papers

The full papers and additional information and material can be found on the workshop's Web site (http://research.microsoft.com/~cszypers/events/WCOP2003/). This site also has the details for the Microsoft Research technical report that gathers the papers and this report.

The following list of accepted papers is sorted by the name of the presenting author.

1. Bobeff, Gustavo; Noyé, Jacques (École des Mines de Nantes, France). "Molding components using program specialization techniques"
2. Dumitrascu, Nicolae; Murphy, Sean; Murphy, Liam (U College Dublin, Ireland). "A methodology for predicting the performance of component-based applications"
3. Falcarin, Paolo; Lago, Patricia; Morisio, Maurizio (Politecnico di Torino, Italy). "Dynamic architectural changes for distributed services"
4. Gybels, Kris (Vrije U Brussel, Belgium). "Enforcing feature set correctness for dynamic reconfiguration with symbiotic logic programming"
5. Ketfi, Abdelmadjid; Belkhatir, Noureddine (Domaine U Grenoble, France). "Dynamic interface adaptability in service oriented software"
6. Kojarski, Sergei; Lorenz, David H. (Northeastern U, USA). "Unplugging components using aspects"
7. Lee, Seung-Yeon; Kwon, Oh-Cheon; Kim, Min-Jung; Shin, Gyu-Sang (Electronics and Telecommunications Research Institute, Korea). "Research on an MDA-based COP approach"
8. Marangozova, Vania (INRIA Rhône-Alpes, France). "Component quality configuration: the case of replication"
9. Matevska-Meyer, Jasminka; Hasselbring, Wilhelm; Reussner, Ralf H. (U Oldenburg, Germany). "Exploiting protocol information for speeding up runtime reconfiguration of component-based systems"
10. Nano, Olivier; Blay-Fornarino, Mireille (Université de Nice-Sophia Antipolis, France). "Using MDA to integrate services in component platforms"
11. Overhage, Sven (Augsburg U, Germany). "Towards a standardized specification framework for component discovery and configuration"

12. Prochazka, Marek (INRIA Rhône-Alpes, France). “A flexible framework for adding transactions to components”
13. Puntigam, Franz (TU Wien, Austria). “State information in statically checked interfaces”
14. Reussner, Ralf H. (U Oldenburg, Germany); Poernomo, Iman H.; Schmidt, Heinz W. (both Monash U, Australia). “Contracts and quality attributes of software components”
15. Thomas, Bill; Vecellio, Gary (The MITRE Corporation, USA). “Infrastructure-based mediation for enforcement of policies in composed and federated applications”
16. Trofin, Mircea; Murphy, John (Dublin City U, Ireland). “A self-optimizing container design for Enterprise JavaBeans applications”

Workshop for PhD Students in Object Oriented Programming

Pedro J. Clemente[1], Miguel A. Pérez[1], Sergio Lujan[2], and Hans Reiser[3]

[1] Department of Computer Science. University of Extremadura, Spain[†]
{jclemente, toledano}@unex.es
[2] Department of Languages and Computer Science. University of Alicante, Spain.
sergio.lujan@ua.es
[3] Department of Distributed Systems and Operating Systems.
University of Erlangen-Nürnberg, Germany.
reiser@cs.fau.de

Abstract. The objective of the 13th edition of Ph Doctoral Students in Object-Oriented Systems workshop (PHDOOS) was to offer an opportunity for PhD students to meet and share their research experiences, and to discover commonalities in research and student ship. In this way, the participants may receive insightful comment about their research, learn about related work and initiate future research collaborations. So, PHDOOS is a gauge to detect hot spots in current lines of research and new avenues of work in objects-oriented concepts.

1 Introduction

At its 13th edition, the PhDOOS workshop established the annual meeting of PhD students in object-orientation. The main objective of the workshop is to offer an opportunity for PhD students to meet and share their research experiences, to discover commonalities in research and studentship, and to foster a collaborative environment for joint problem solving.

The workshop also aims at strengthening the International Network of PhD Students in Object-Oriented Systems[17] initiated during the 1st edition of this workshop series at the European Conference on Object Oriented Programming (ECOOP), held in Geneva, in 1991. This network has counts approximately 120 members from all over the world. There is a mailing list and a WWW site, used mainly for information and discussion about OO-related topics. Since its foundation, the International Network of PhD Students has proposed and created a workshop for PhD students in association with ECOOP each year. This 13th edition makes PHDOOS a classical workshop in ECOOP.

At this edition, the workshop was divided into plenary sessions, discussion session and a conference. The sessions were determined according to the research interests of the participants. Potential topics of the workshop were those of the

[†] The organization of this workshop has been partially financed by CICYT, project number TIC02-04309-C02-01

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 50–61, 2004.

main ECOOP conference, i.e. all topics related to object technology including but not restricted to: analysis and design methods, real-time, parallel systems, patterns, distributed and mobile object systems, aspects oriented programming, frameworks, software architectures, software components, reflection, adaptability, reusability and theoretical foundations. Due to the heterogeneous nature of the topic of the papers received, the workshop conclusions are focused on the interesting research areas and on solving common problems.

The participants had a 20 minute presentation at the workshop (including questions and discussions). The discussion group was based on the research interests of the participants. Finally, the conference featured a speaker who is invited to talk about interesting research, personal experiences or research methodology. This conference was a unique opportunity to hear or ask things not discussed elsewhere, and to have an "unplugged" discussion with a well-known personality from our field. This year, the speakers was the professor Robert E. Filman

This paper is organized into the following points. The next section is focused on the participants of the workshop. In section three, the presented papers are explained and main topics are summarized. The fourth section talks about a conferences, and the fifth section includes the final discussion. Finally the paper concludes with workshop conclusions and bibliographic references.

2 PHDOOS Workshop Participants

In previous editions [4,1,2,3] of this workshop there have been more or less twenty participants working for 2 days. However, this year the coincidence with a Doctoral Symposium cut the number of participants with papers to 10, thereby reducing the length of this workshop to 1 day. We can divide the participants into four groups:

- Participants with papers
- Organizing Committee.
- Program Committee
- Invited speaker.

2.1 Participants with Papers

The number of papers received was 11, and 9 were accepted. The attendants with accepted papers were the following:

- M. Devi Prasad. Manipal Center for Information Science, Manipal Academy of Higher Education.
- Hervé Paulino. Department of Computer Science. Faculty of Sciences and Technology. New University of Lisbon.
- Sari R. ElDadah, Nidal Al-Said. Department of Information Systems. Arab Academy For Banking and Financial Sciences.
- Ademar Aguiar. Faculdade de Engenharia da Universidade do Porto.

- Balázs Ugron. Department of General Computer Science. Eötvös Loránd University, Budapest.
- Amparo Navasa Martínez. Quercus Software Engineering Group. University of Extremadura.
- Susanne Jucknath. Institute of Software Engineering and Theoretical Computer Science Technical University of Berlin.
- Jan Wloka. Fraunhofer FIRST.
- Szabolcs Hajdara. Department of General Computer Science. Eötvös Loránd University, Budapest.
- Andreas I. Schmied. Distributed Systems Laboratory. University of Ulm.

2.2 Organizing Committee

The Organizing Committee of Ph Doctoral Object-Oriented Systems is made up of volunteers participants in the previous workshop. Organizers in earlier editions advise these volunteers. This year, the organizers were: Sergio Luján Mora, Sérgio Soares, Hans Reiser, Pedro José Clemente Martín and Miguel Angel Pérez Toledano.

Sergio Luján-Mora is a Lecturer at the Computer Science School at the University of Alicante, Spain. He received a BS in 1997 and a Master in Computer Science in 1998 from the University of Alicante. Currently he is a doctoral student in the Dept. of Language and Information Systems being advised by Dr. Juan Trujillo. His research spans the fields of Multidimensional Databases, Data Warehouses, OLAP techniques, Database Conceptual Modeling and Object Oriented Design and Methodologies, Web Engineering and Web programming.

Sérgio Soares is currently a Ph.D. student at the Computer Science Center of the Federal University of Pernambuco. His current research interests include implementation methods and aspect-oriented programming. He is also a Assistant Professor at the Statistics and Computer Science Department of the Catholic University of Pernambuco

Hans Reiser studied Computer Science at the University of Erlangen-Nürnberg, obtained Dipl. Inf. degree in computer science in spring of 2001. Since 6/2001 he has been employed as researcher at the Department of Distributed Systems and Operating Systems at University of Erlangen-Nürnberg. He is member of the AspectIX research team (a joint group of our department and distributed systems department of University of Ulm), doing research on adaptive middleware for large-scale distributed systems. The primary research focus for PhD is software based fault tolerance in distributed systems.

Pedro José Clemente Martín graduated with a Dipl. Inf. in Computer Sciences from the University of Extremadura (Spain) in 1998.He is a Lecturer at the Computer Science Department at the University of Extremadura, Spain and a member of the Quercus Software Engineering Group. His current research interests are Aspect Oriented Programming and Component based Software Engineering. His PhD is focused on the interconnection of components to build software systems using aspect oriented programming.

Miguel Angel Pérez Toledano graduated with a Dipl. Inf. in Computer Sciences from the Polithecnic University of Cataluña (Spain) in 1993. Now, he is working at the University of Extremadura as Associated Professor. He participated in the organization of the PhD workshop of ECOOP'02, and is a member of the Quercus Software Engineering Group. His current research interests are Semantic Description of Components and Repositories of Software Components. His PhD is focused on the selection and retrieval of components from repositories using semantic descriptions.

2.3 Program Committee

The program committee was composed of senior researchers with a strong background in some object-oriented topic. The review process is designed to ensure that every participant is able to present some relevant and well prepared material. This edition, the program committee was composed by:

- Marcelo Faro do Amaral Lemos (Federal University of Pernambuco, Brazil)
- Márcio Lopes Cornelio (Federal University of Pernambuco, Brazil)
- Juan C. Trujillo (University of Alicante, Spain)
- Fernando Sánchez Figueroa (University of Extremadura, Spain)
- Juan Manuel Murillo Rodríguez (University of Extremadura, Spain)
- Rüdiger Kapitza (University of Erlangen-Nürnberg, Germany)
- Andreas Schmied (Univiversity of Ulm, Germany)
- José Samos (University of Granada, Spain)

2.4 Invited Speaker

For this edition, the invited speaker to PhDOOS was professor Robert E. Filman. He is working at the Research Institute for Advanced Computer Science (RIACS), NASA Ames Center. His work about Aspect Software Oriented Development is recognized by the international computer science research community.

3 Research Hot Points

One of the main workshop objectives is to detect the principal research lines following Object Oriented Programming and Programming Technologies. In this sense, the following topics have been presented and have been discussed:

- Aspect Oriented Software Development
- Documentation and Categorization of Software
- Agents Technologies

3.1 Aspect Oriented Software Development

Aspect Oriented Software Development has been the most important point discussed in this workshop due to the fact that sixty percent of the accepted papers dealt with this topic. In this sense, we can find several topics and focuses that allow us to ensure that the area of Aspect Oriented Technologies is currently very important.

The subjects treated included extension and description of Aspect-Oriented Languages, Aspect Oriented Code Visualization, Composition of Aspects, and domain of Aspect-Oriented application.

About *Aspect-Oriented Languages* there are two approaches: to extend an existing language (for example, AspectJ) and to define new languages to describe and compose aspects (for example, using ADL).

Prassat [13] suggests that AspectJ does not treat type-casting as an important operation in programs execution. He demonstrates that Java programs use type casting primarily for retrieving references to new types from distantly related ones. He investigates AspectJs inadequacy in selecting execution points that use type-casting to yield new object or inheritance references. He demonstrates the necessity for a reference creation by type casting joinpoints and argues that its addition makes AspectJs existing model more expressive.

Amparo Navasa [6] claims that to extract the aspect code, crosscutting the functional one makes it easier to implement aspect oriented systems using AOP languages, but it is possible to take away the problem of AO from the implementation phase to the design. In this phase, crosscutting functional code concerns can be treated as independent entities. Her ideas are based on developing aspect oriented systems taking into account the benefits of applying CBSE at the early stages of AO systems development, particularly at architectural design. Concretely, AOSD at the design level as a co-ordination problem, using an ADL to formalize it. This means a new language to define aspects and compose them from the design time viewpoint.

Aspect Oriented Software Visualization presents a growing potential due to the fact that graphic tools do not currently exist to visualize the aspect code execution. In this sense, one intention of Software Visualization is to form a picture in the user's mind of what this software is about, what happens during the execution and what should happen from the programmer's point of view[12]. This is especially helpful with existing software and non-existing documentation. Therefore the amount of existing software is increasing the need to understand this software too.

From *Composition of Aspects* viewpoint, there is special interest in Distributed Systems, because they contain lots of cross-cutt and tangled code fragments to fulfill their services. Merging a formerly separated aspect code with a program code by means of aspect-oriented programming is enabled through a couple of available technologies that manipulate program sources. Unfortunately, these tools or aspect weavers both operate on a distinct coarse-grained level (types, methods) and fulfill only a restricted a-priori known set of manipulations.

However, to weave several aspect code fragments which could have been constructed by independent teams for more than one concern simultaneously a composition that not only concatenates aspects, but also manages join effects between them, reveals several complex, possibility interfering weaving demands[7].

The use of *Aspect Oriented* in specific domains or concrete areas in Software Engineering is other growing area. In this sense, Hajdara[10] presented a solution to apply Aspect Oriented technologies to handle different non-functional properties like synchronization specification of parallel systems.

Refactoring and Aspect Orientation (AO) are both concepts for decoupling, decomposition, and simplification of object-oriented code. Refactoring is meant to guide the improvement of existing designs. For this reason it is the main practice in eXtreme Programming to implement 'embrace change' in a safe and reliable way. Aspect orientation on the other hand offers a new powerful encapsulation concept especially for coping with so called crosscutting concerns. Although refactoring and AO have the same goals, their current forms impede each other. Since the development of modular systems has become more and more difficult a combined application of refactoring and AO is still a desirable goal and would be a great help for developers[16].

3.2 Documentation and Categorization of Software

Two papers were presented on this topic during the workshop. The first paper "A Process-based Framework for Automatic Categorization of Web Documents" from Sari R. ElDadah[15], presented the design of a Framework for the development of Automatic Text Categorization applications of Web Documents. The process, composed of 4 activities, (identifying significant categories, finding the best description for each category, classifying the documents into the identified categories and personalizing the categories and their relevant descriptions according to the user preference) is conducted from the various Automatic Text Categorization methods developed so far, and described based on Petri Nets process description language. The paper concluded with notes about the ATC process.

The second paper presented on this topic was titled "A Minimalist Approach to Framework Documentation" from Ademar Aguiar and Gabriel David[5]. This paper proposes a documenting approach for frameworks that aims to be simple and economical to adopt. It reuses existing documentation styles, techniques and tools and combines them in a way that follows the design principles of minimalist instruction theory. The minimalist approach proposes a documentation model, a documentation process, and a set of tools built to support the approach (Extensible Soft Doc).

3.3 Agents Technologies

In this subject, we can include the paper "Mob: a Scripting Language for Programming Web Agents" showed by Hervé Paulino[11]. Mob: a scripting language for programming mobile agents in distributed environments was presented. The

semantics of the language is based on the DiTyCO (Distributed TYped Concurrent Objects) process calculus. Current frameworks that support mobil agents are mostly implemented by defining a set of Java classes that must then be extended to implement a given agent behavior. Mob is a simple scripting language that allows the definition of mobile agents and their interaction, an approach similar to D'Agents. Mob is compiled into a process-calculus based kernel-language, and its semantics can be formally proved correct relative to the base calculus.

4 Workshop Conference

The workshop conference entitled *Aspect Oriented Software Development* was presented by Robert E, Filman.

Robert presented an interesting conference which has four parts:

- Object Infrastructure Frameworks (OIF)
- Aspect-Oriented Software Development
- AOP through Quantification over Events
- Research Remarks

The conference began with an introduction about Robert's latest research projects. He then introduced the concept of *Aspect Oriented Software Development,* and a way to obtain AOP through Quantification over Events. Finally, some remarks about research directions were presented.

Now, we are going to summarize each part of this presentation, because Robert's affirmations are very interesting, above all for students interested in Aspect-Oriented.

4.1 Object Infrastructure Framework (OIF)

Distributed Computing systems is difficult mainly for the following reasons:

- It is hard to archive systems with systematic properties (called *Ilities*) like *Reliability, Security, Quality of Service, or Scalability.*
- Distribution is complex for the following reasons: concurrence is complicated, distributed algorithmics are difficult to implement, every policy must be realized in every component, frameworks can be difficult to use, etc.

The introduction of a Component based Architecture require separating the component functionality and the non-functional properties. These non-functional properties should be inserted into components and allow for the interaction (communications) among components.

This idea has been implemented using Object Infrastructure Frameworks (OIF) [14]. OIF allows for the injection of behavior on the communications paths between components, using *injectors* because they are discrete, uniform objects, by object/methods and dynamically configurable. This idea permit the implementation of non-functional properties like *injectors*, and then they can be

applied to the components; for example, injectors can encrypt and decrypt the communications among components.

OIF is an Aspect Oriented Programming mechanism due to the fact that:

- It allows separating concerns into injectors
- It wrapping technology
- It piggy-backs on distributed-object technology (CORBA)

4.2 Aspect-Oriented Software Development

How can we structure our programming languages do help us archive such ilities(Reliability, Security, Quality of Services, Evolvability, etc.)?

Separation of Concerns is an interesting strategy to structure our programming languages because a fundamental engineering principle is that of *separation of concerns.*

Separation of Concerns promises better maintainability, evolvability, Reusability and Adaptability. Concerns occur at both the User/requirements level and Design/implementation level[8].

Concerns cross-cut can be Applied to different modules in a variety of places, and must be composed to build running systems.

In conventional programming, the code for different concerns often becomes mixed together (tangled-code).

Aspect Oriented Programming modularize concerns that would otherwise be tangled. AOP provides mechanisms to weave together the separate concerns.

Implementation Mechanism. The following division allows for the description of the common AOP implementation mechanisms used and the usual platforms used:

- *Wrapping technologies*: Composition filters, JAC
- *Frameworks*: Aspect-Moderator Framework
- *Compilation technologies:* AspectJ, HyperJ
- *Post-processing strategies*: JOIE, JMangler
- *Traversals*: DJ
- *Event-based*: EAOP
- *Meta-level strategies*: Bouraqadi et al., Sullivan, QSOUL/Logic Meta-Programming

4.3 AOP Through Quantification over Events

A single concerns can be applied to many places in the code, but the we need to quantify it.

Concerns can be quantified over the static(lexical) form of the program, semantic (reflective) structure of the program structures and the events that happen in the dynamic execution of a system.

To take the expressiveness in quantification to its extreme is to be able to quantify over all the history of events in a program execution. The events are with respect to the abstract interpreter of a language. However, language definitions do not define their abstract interpreters.

As a consequence, we are able to describe interesting points in the program (lexical structure of the program, reflective structure of the classes and dynamic execution of the system), and then to describe the change in behavior desired at these points. The shadow of a description is the places in the code where the description might happen, for example, the event *invoking a subprogram* represents in a syntactic expression *subprogram calls.* It is necessary to define these events, capture these, and to change the behavior at this point. For more detail, please refer to [9].

4.4 Research Remarks

This section of the conference presents the main research directions about *Aspect Oriented Software Development*, and this information should be useful for current and prospective Phd Students.

Research Regime

- Define a language of events and actions on those events.
- Determine how each event is reflected (or can be made visible) in source code.
- Create a system to transform programs with respect to these events and actions.
- Developing an environment for experimenting with AOP languages (DSL for AOP)

Real AOP Value

- We don't have to define all these policies before building the system
- Developers of tools, services, and repositories can remain (almost) completely ignorant of these issues
- We can change the policies without reprogramming the system
- We can change the policies of a running system

Open Research Directions

Languages for Doing AOP

- Hardly seen the end of this topic
- Join points
- Weaving mechanisms
- Handling conflicts among aspects

The Software Engineering of AOP Systems

- Modeling aspects: From models to executable code
- Debugging aspects independently of the underlying system
- Tools for recognizing and factoring concerns

Applying AOP to Particular Tasks

- Monitoring/debugging
- Version control/provenance
- Web/system services
- User-centered computing
- Reliable systems
- System management

5 Summary of the Discussion Group

Although the workshop has a wide focus it turned out that most participants are working in research areas closely related to aspect oriented programming. Instead of having small subgroup discussions, the group opted for one plenary sessions discussing topics on AOP-related topics. In the process of selecting appropriate topics, we came up with the following main issues:

5.1 Shall Aspects Be Supported as First Class Entities?

The situation today is that most AOP languages do not support aspects as first class entities. This is however due to the simple pragmatic way these languages are implemented. One may anticipate that the next generation of AOP languages will provide support for aspects as first class entities. The main benefits which we expect from such future developments are reusability of aspects, inheritance between aspects, and dynamically adaptive aspects. These areas still offer some research potential.

5.2 Does AOP Make Sense with Non-OOP Paradigms?

This issue was only briefly discussed. The group agreed that in paradigms like functional or imperative programming, separation of concerns is an equally required design method, and AOP is useful technique to support this. However, related to the generalization of aspects discussed later, non-OOP aspect orientation will have somewhat different requirements on potential join-point definition than in the case of OOP.

5.3 What Are Adequate Techniques to Understand AOP Programs?

One major problem with AOP is that while it simplifies things on a rather abstract level, it gets more difficult to understand the concrete behavior of your program at a lower level. Current visualization techniques, already offered in some development environments, are not yet adequate for larger projects. The issues to be supported by visualization techniques are documentation, testing and debugging. The demand for such techniques will further rise, if aspect code and functional code shall be developed independently

5.4 What Purposes Shall Aspects Be Used For?

One widespread use of aspects is to restructure existing software (refactoring to increase modularization), with the goal to improve structure and maintainability. However, we anticipate that in the future, AOP will also be applied for new applications, starting in the design phase. For this purpose, the availability of reusable "aspect components", addressed in the next item, will be essential. A different question is whether AOP techniques may and shall be used for modifying existing applications, that were developed without considering such later modification. However, we were rather reluctant to consider this as a good AOP technique.

5.5 Is Is Feasible to Make Aspects Reusable?

Closely related to the previous topic is this important issue. In our opinion, the most problematic matter is the definition of join points. In current AOP languages, the definition of aspects is always tailored individually to one specific application. Even in such a specific case, existing AOP tools are usable best if the application and the aspects are written by the same developer. Also, even a small modification to one application easily makes aspects unusable. We all agree that this is a highly disappointing situation.

Having reusable aspects is highly desirable, but it requires further research on how this might be done. An important issue in this context is the question of whether aspects can be defined without them? limiting to an specific AOP language. Ultimately, AOP needs to be done already in the design phase of application development.

5.6 Conclusions

In spite of the fact that AOP has matured for over the years, several issues can be found that are still relevant for future research. The most important issue we found are the definition of join points targeting at reusability of aspects, and tool support for visualizing and understanding aspect oriented applications.

References

1. 10th Workshop for Ph Doctoral Students in Objects Oriented Systems. http://people.inf.elte.hu/phdws/, 2000.
2. 11th Workshop for Ph Doctoral Students in Objects Oriented Systems. http://www.st.informatik.tu-darmstadt.de/phdws/, 2001.
3. 12th Workshop for Ph Doctoral Students in Objects Oriented Systems. http://www.softlab.ece.ntua.gr/facilities/public/AD/phdoos02/, 2002.
4. 9th Workshop for Ph Doctoral Students in Objects Oriented Systems. http://www.comp.lancs.ac.uk/computing/users/marash/PhDOOS99/, 1999.
5. Gabriel David Ademar Aguiar. A minimalist approach to framework documentation. *13th Workshop for Phd Students in Object Oriented Programming at ECOOP. Darmstadt, Germany.*, 2003.
6. J.M. Murillo A.Navasa, M.A.Pérez. Using an adl to design aspect oriented systems. *13th Workshop for Phd Students in Object Oriented Programming at ECOOP. Darmstadt, Germany.*, 2003.
7. Franz J. Hauck Andreas I. Schmied. Composing non-orthogonal aspects. *13th Workshop for Phd Students in Object Oriented Programming at ECOOP. Darmstadt, Germany.*, 2003.
8. Robert E. Filman and Daniel P. Friedman. Aspect-oriented programming is quantification and obliviousness. *Workshop on Advanced Separation of Concerns. OOPSLA, Minneapolis*, 2000.
9. Robert E. Filman and Klaus Havelund. Source-code instrumentation and quantification of events. *Workshop on Foundations Of Aspect-Oriented Languages (FOAL) at AOSD Conference. Twente, Netherlands.*, 2002.
10. Szaboles Hajdara. An example of generating the synchronization code of a system composed by many similar objects. *13th Workshop for Phd Students in Object Oriented Programming at ECOOP. Darmstadt, Germany.*, 2003.
11. Luís Lopes Herve Paulino and Fernando Silva. Mob: a scripting language for programming web agents. *13th Workshop for Phd Students in Object Oriented Programming at ECOOP. Darmstadt, Germany.*, 2003.
12. Susanne Jucknath. Software visualization and aspect-oriented software development. *13th Workshop for Phd Students in Object Oriented Programming at ECOOP. Darmstadt, Germany.*, 2003.
13. M. Devi Prasad. Typecasting as a new join point in AspectJ. *13th Workshop for Phd Students in Object Oriented Programming at ECOOP. Darmstadt, Germany.*, 2003.
14. Diana D. Lee Robert E. Filman, Stu Barrett and Ted Lindero. Iserting ilities by controlling communications. *Communications of the ACM, January*, 45, No 1:118–122, 2002.
15. Nidal Al-Said Sari R. ElDadah. A process-based framework for automatic categorization of web documents. *13th Workshop for Phd Students in Object Oriented Programming at ECOOP. Darmstadt, Germany.*, 2003.
16. Jan Wloka. Refactoring in the presence of aspects. *13th Workshop for Phd Students in Object Oriented Programming at ECOOP. Darmstadt, Germany.*, 2003.
17. International Network for PhD Students in Object Oriented Systems (PhDOOS). http://www.ecoop.org/phdoos/, 1991.

Formal Techniques for Java-Like Programs

Susan Eisenbach, Gary T. Leavens, Peter Müller,
Arnd Poetzsch-Heffter, and Erik Poll

se@doc.ic.ac.uk, leavens@cs.iastate.edu, peter.mueller@inf.ethz.ch,
poetzsch@informatik.uni-kl.de, erikpoll@cs.kun.nl

Abstract. This report gives an overview of the fifth ECOOP Workshop on Formal Techniques for Java-like Programs. It explains the motivation for such a workshop and summarizes the presentations and discussions.

1 Introduction

This workshop was the fifth in the series of workshops on "Formal Techniques for Java-like Programs (FTfJP)" held at ECOOP. It was a follow-up to FTfJP workshops held at the previous ECOOP conferences in 2002 [6], 2001 [5], 2000 [2], and 1999 [4], and the "Formal Underpinnings of the Java Paradigm" workshop held at OOPSLA'98.

The workshop was organized by

- Susan Eisenbach (Imperial College, Great Britain),
- Gary T. Leavens (Iowa State University, USA),
- Peter Müller (ETH Zurich, Switzerland),
- Arnd Poetzsch-Heffter (University of Kaiserslautern, Germany), and
- Erik Poll (University of Nijmegen, the Netherlands).

The program committee of the workshop included

- John Boyland (University of Wisconsin, USA),
- Gilad Bracha (Sun Microsystems, USA),
- Alessandro Coglio (Kestrel Institute, USA),
- Sophia Drossopoulou (Imperial College, UK),
- Doug Lea (State University of New York at Oswego, USA),
- Gary T. Leavens (Iowa State University, USA),
- K. Rustan M. Leino (Microsoft Research, USA)
- Peter Müller, Chair (ETH Zurich, Switzerland),
- David Naumann (Stevens Institute of Technology, USA),
- Tobias Nipkow (Technische Universität München, Germany),
- James Noble (Victoria University of Wellington, New Zealand),
- Erik Poll (University of Nijmegen, the Netherlands).
- Don Syme (Microsoft Research, UK).

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 62–71, 2004.

The proceedings of the workshop has appeared as technical report [3] and is available on the web at
`www.inf.ethz.ch/research/publications/show.php?what=408`.

There was lively interest in the workshop. We were very pleased with the high quality of the submissions. Out of 24 submissions, 16 papers were selected by the Program Committee for longer presentations. In addition, for one position paper a shorter presentations was given. 38 people from 11 countries attended the workshop.

Motivation. Formal techniques can help to analyze programs, to precisely describe program behavior, and to verify program properties. Applying such techniques to object-oriented technology is especially interesting because:

- the OO-paradigm forms the basis for the software component industry with their need for certification techniques,
- it is widely used for distributed and network programming,
- the potential for reuse in OO-programming carries over to reusing specifications and proofs.

Such formal techniques are sound, only if based on a formalization of the language itself.

Java is a good platform to bridge the gap between formal techniques and practical program development. It plays an important role in these areas and is becoming a de facto standard because of its reasonably clear semantics and its standardized library.

However, Java contains novel language features, which are not fully understood yet. More importantly, Java supports a novel paradigm for program deployment, and improves interactivity, portability and manageability. This paradigm opens new possibilities for abuse and causes concern about security.

Thus, work on formal techniques and tools for Java programming and formal underpinnings of Java complement each other. This workshop aims to bring together people working in these areas, in particular on the following topics:

- specification techniques and interface specification languages,
- specification of software components and library packages,
- automated checking and verification of program properties,
- verification technology and logics,
- Java language semantics,
- dynamic linking and loading, security.

With the advent of languages closely related to Java, notably C#, we have changed to title to "Formal Techniques for Java-*like* Programs" to also include work on these languages in the scope of the workshop.

Structure of Workshop and Report. The one-day workshop consisted of a technical part during the day and a workshop dinner in the evening. The presentations at the workshop were structured as follows:

9:00-10:30 Session 1: Language Semantics (Chair: Gilad Bracha)

- *Inner Classes visit Aliasing*
  Matthew Smith and Sophia Drossopoulou
- *Java Definite Assignment in Isabelle/HOL*
  Norbert Schirmer
- *Flexible, source level dynamic linking and re-linking*
  Sophia Drossopoulou
- *Algebraic Semantics of the Statements of Sequential Java*
  Kazem Lellahi and Alexandre Zamulin

11:00-12:30 Session 2: Specification and Verification (Chair: Arnd Poetzsch-Heffter)

- *JML Support for Primitive Arbitrary Precision Numeric Types: Definition and Semantics*
  Patrice Chalin
- *Verifying JML specifications with model fields*
  Cees-Bart Breunesse and Erik Poll
- *Verification of Object-Oriented Programs with Invariants*
  Mike Barnett, Robert DeLine, Manuel Fähndrich, K. Rustan M. Leino, Wolfram Schulte
- *A Tool-supported Assertional Proof System for Multithreaded Java*
  Erika Ábrahám, Frank S. de Boer, Willem-Paul de Roever, Martin Steffen

13:30-15:00 Session 3: Refinement and Static Analysis (Chair: Erik Poll)

- *Toward Automatic Generation of Provably Correct Java Card Applets*
  Alessandro Coglio
- *Ownership: transfer, sharing, and encapsulation*
  Anindya Banerjee and David A. Naumann
- *Static Detection of Atomicity Violations in Object-Oriented Programs*
  Christoph von Praun and Thomas R. Gross
- *Checking Concise Specifications for Multithreaded Software*
  Stephen Freund and Shaz Qadeer
- *Safety and Security through Static Analysis (Short presentation)*
  Rene Rydhof Hansen and Igor Siveroni

15:30-17:00 Session 4: Language Implementation and Runtime Checking (Chair: John Boyland)

- *Stronger Typings for Separate Compilation of Java-like Languages*
  Davide Ancona and Giovanni Lagorio
- *Static analysis for eager stack inspection*
  Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari
- *Abstraction-carrying Code: A New Method to Certify Temporal Properties*
  Songtao Xia and James Hook
- *Instrumentation of Java Bytecode for Runtime Analysis*
  Allen Goldberg and Klaus Havelund

2 Session 1: Language Semantics

Formalizations of Java's language semantics are the basis for analyzing both language properties and Java programs. The first session comprised presentations of formalizations of Java and discussed possible extensions. It started with two presentations of formalizations of parts of the Java programming language: inner classes and definite assignment.

Matthew Smith's and Sophia Drossopoulou's paper presents an operational semantics and a type system for a Java subset with inner classes and aliasing. Based on this formalization, it states several interesting properties such as soundness in the presence of aliasing. Moreover, it shows that a program with inner classes and its translation into a program without inner classes are semantically equivalent. Formalizing inner classes provided interesting insights into the rationale of some of the restrictions imposed by the language.

Norbert Schirmer presented the formalization of another aspect of Java, Java's rules for the definite assignment. These rules guarantee that variables are initialized before they are used. Therefore, definite assignment is a prerequisite for many interesting language properties such as type safety. Schirmer showed that von Oheimb's type safety proof, which assumes that local variables are set to default values when they are declared, can be generalized to definite assignment. Especially, the definite assignment rules are strong enough to guarantee type safety. The formalization is developed in Isabelle/HOL for a large subset of Java including abrupt termination.

In contrast to the first two presentations, Sophia Drossopoulou's and Susan Eisenbach's work on dynamic linking and re-linking goes beyond current Java or C# implementations. Besides supporting the loading and verification of classes interleaved with program execution, it also allows type-safe removal and replacement of classes, fields, and methods. Such extended features support unanticipated software evolution, that is, updates during program execution. Sophia Drossopoulou presented the semantics of a language similar to Java and C# with extended dynamic linking. The model is purely at the source language level, which is necessary since the effects of dynamic linking are visible to the source language programmer.

Alexandre Zamulin presented an algebraic semantics of statements of sequential Java. The semantics is defined in the style of algebraic specifications: The text of a program is represented as an algebraic specification and the semantics of the program is defined as a set of models of the specification. The novelty of this approach consists in defining a specialized high-level operational machine whose components naturally represent program components. The work is based on Abstract State Machines and D-oids. Extending the formalization to concurrent programs is planned as future work.

3 Session 2: Specification and Verification

Specification technologies enable to state behavioral properties of programs in a formal way. They are important for precise documentation of what programs

are expected to do. In addition, they are the prerequisite for verification, i.e. for developing mathematical and formal proofs that a program has a certain property. This session presented and discussed techniques

- to handle arbitrary precision numeric types in a specification language,
- to do verification in the presence of abstraction,
- to formulate and prove object invariants, and
- to verify multithreaded programs.

The following paragraphs introduce these aspects and summarize the main results of the presentations.

The Java Modeling Language JML is a behavioral specification language for Java. It uses Java expressions in pre- and postconditions. Consequently, the numerical types used in these expressions are finite like in Java and have modulo arithmetic (e.g. adding one to the maximal integer of Java yields its minimal integer). This often leads to misinterpretations of specifications, because many software developers expect that specifications are based on infinite arithmetic. In the first contribution of the session, Patrice Chalin motivated the problem of finite arithmetic in specification languages and presented an approach to smoothly extend JML and its type system by arbitrary precision arithmetic. In particular, he showed how finite and infinite types can coexist so that the Java types can still be used where necessary.

To express program properties, one often wants to use notions that are more abstract than those used in a program itself. For example, abstract data types like sets, lists, etc. can be very helpful in specifying program behavior. JML supports such abstraction by so-called model fields. These model fields can range over abstract domains. To relate a class or interface that is specified using model fields to its implementation, JML supports the specification of representation functions and relations. A representation function defines the value of a model field in terms of concrete field values, a representation relation establishes a property that the value of a model field has to satisfy in relation to concrete field values. In the second contribution of the session, Cees-Bart Breunesse and Erik Poll investigated approaches to verify model fields with the help of the LOOP tool. After a motivating discussion, they proposed two solutions by desugaring representation relations into a JML subset without such relations.

The basic language constructs for the specification of object-oriented programs are pre- and postconditions and object invariants (or similarly class invariants). Pre- and postconditions describe the behavior of methods. Object invariants state consistency constraints on the data stored in one object and on the relation between objects. Although the importance of object invariants is recognized for quite a while in the research community, there does not exist a standard semantics and generally excepted application methodology for object invariants. In the third contribution of the session, K. Rustan M. Leino presented a new approach to object invariants. The approach combines ideas from ownership models and uniqueness with an explicit specification discipline of when fields may be written and invariants be temporarily violated. It is suf-

ficiently fine-grained to expose only parts of the inherited fields and invariants to modification. It allows for modular verification.

So far, research on verification of Java programs concentrated on its sequential kernel with constructs for dynamic object creation, subtyping, inheritance, and dynamic binding. The fourth contribution of the session, presented by Erika Ábrahám, was about an assertional proof system for a Java subset with multithreading (leaving out e.g. features like inheritance, subtyping, and exceptions). The work extends Owicki-Gries-style proof outlines with logical techniques developed for the verification of communicating sequential processes and adapts the result to the special requirements of Java. The provided assertion language allows for the specification of safety properties. Program verification is supported by the Verger tool that generates verification conditions from an annotated program. The remaining verification task is to prove the resulting conditions within the theorem prover PVS.

4 Session 3: Refinement and Static Analysis

Java Card, the Java dialect for programming smart cards, is a very interesting target for formal techniques for Java, if only because it does not have the complications of multithreading discussed in the last talk of Session 2. Alessandro Coglio gave an overview of work on the automatic generation of Java Card applets from high level specifications. The aim of this work is to generate code form high level specifications. This is done by automatically translating the high level specification, written in a domain-specific language, via a series of refinement steps to more concrete specifications, and finally producing Java Card code from the most refined specification. The approach uses the Specware system developed at the Kestrel Institute, via a shallow embedding of Java Card in the logic of Specware. As for every refinement step an associated proof is generated, the code is generated together with a proof of its correctness that could be independently checked.

For Java, as for all imperative languages, one of the main difficulties in reasoning about programs is the existence of a global heap, which contains a complicated pointer structure which gives rise to complications such as aliasing or the leaking of references. (Indeed, there was another ECOOP workshop, WS 21, entirely dedicated to this topic, the First International Workshop on Aliasing, Confinement and Ownership in Object-oriented Programming (IWACO), and David gave a repeat performance of his talk there the day following FTfJP.) David Naumann talked about his work on the notion of 'ownership' as a way to express encapsulation of heap structures and as a means to support modular reasoning. He presented a generalization of earlier work which supported multiple ownership and allowed ownership to be transferred from one object to another.

The next two talks both dealt with static analyses for multi-threaded programs.

Christoph von Praun talked about an analysis to detect possible errors in multi-threaded programs, namely violations of method-level atomicity. Such er-

rors occur when access to shared data is not correctly regulated through the use of locks, so that the effect of a method execution becomes dependent on the execution order of other, concurrent, threads. The analysis is based on a notion of 'method consistency' which depends of the usage of locks to control access to shared data. Violations of this notion of method consistency often coincide with typical synchronization errors, so checking for method consistency is a good way of detecting bugs. A whole program analysis for method consistency has been implemented and has been shown to be efficient enough to be practical.

Stephen Freund also talked about an analysis to improve the reliability of multi-threaded programs. Central motivation for his work was the question of how to achieve more modular verification. To allow thread-modular verification, his approach relied on the use of so-called access predicates; an access predicate specifies, for a given shared variable, when a thread may access it. To allow method-modular verification, his approach relied on a specification per method, related to the implementation by an abstraction relation. A case study showed that for complex multi-threaded systems the requirements could be concisely specified in this framework, and that automated verification of important properties, using the Calvin-R static checker, was feasible.

Last talk in the session returned to the subject of Java Card. In a short talk Igor Siveroni sketched the static analysis for Java Card programs that has been developed in the SecSafe project. Unlike the previous talks, the analysis is not aimed at detecting errors due to the multi-threading. Instead, this analysis, consisting of a 'traditional' control flow and data flow analysis, is aimed at ensuring certain safety and security properties that are important for Java Card applets. The analysis is built on a formal operational semantics of the Java Card Virtual Machine Language. Properties that can be analyzed include the absence of certain exceptions due to runtime errors (e.g., division by zero, dereferencing of null pointers or accessing arrays outside bounds, violations of the Java Card firewall), the leaking of confidential information, and properties about pointer confinement.

5 Session 4: Language Implementation and Runtime Checking

The paper "Stronger Typings for Separate Compilation of Java-like Languages" by Davide Ancona and Giovanni Lagorio addressed the semantics of separate compilation in Java-like languages. Building on Cardelli's formal framework for treating separate compilation [1], they compare the strengths of type environments. A stronger type environment can type check anything that a weaker type environment could. The key problem is to find the strongest type environment that is compatible with the modules being checked. Such a type environment allows one to type check individual modules with the least amount of recompilation needed when one of them changes. Results in the paper include a compositional type system that is able to find the strongest (principal) typing for modules. In this sense it does better than the Java SDK at preventing unnecessary recompilations.

The paper "Static analysis for eager stack inspection" by Massimo Bartoletti, Pierpaolo Degano, and Gian Luigi Ferrari addressed static analysis questions related to security issues. More specifically, the paper treats optimization of Java's technique of stack inspection, which the JVM uses to make access control decisions. The JVM uses a lazy strategy for stack inspection; that is (p. 1), "the call stack is retrieved and inspected only when ... access control is performed." This lazy strategy slows execution and prevents optimizations that would affect stack management. The static analysis described in the paper permits optimization and dynamic linking. It allows for fast context updates. The modular analysis described in the paper is presented formally, and a soundness and completeness result is sketched.

James Hook presented a paper titled"Abstraction-carrying Code: A New Method to Certify Temporal Properties", whose first author is Songtao Xia. This paper addressed the certification of temporal properties, such as liveness. The overall problem is to make the idea of proof carrying code work for questions other than safety properties. Their approach, called "abstraction carrying code," is to let the program carry not a proof but an abstraction. In particular the program carries an abstract interpretation that would be useful in model checking (of various temporal properties). This allows model checking to proceed much more quickly by side-stepping the exponential growth in the size of the state space that would occur without it. There is a type system that ensures that this abstract interpretation matches the code. Several examples were discussed.

The paper "Instrumentation of Java Bytecode for Runtime Analysis" by Allen Goldberg and Klaus Havelund addresses the problem of run-time checking of temporal logic assertions. Because the authors are interested in timed temporal logics, a major problem is how to do this checking without interfering with the timing of the program being run. The solution described is a tool, JSpy, that instruments Java byte codes so that they produce an event stream which can be analyzed off-line after the program has run. They described various tradeoffs between making the instrumented byte code do more work or making the observer of the event stream do more work; JSpy itself is designed to have the instrumented byte code do as little as possible. The techniques seem also to have applications to real-time monitoring, as the architecture permits different observers to be plugged in dynamically to the event stream of the program.

6 Conclusions

Looking back on the workshop, the program committee is very pleased with the quality of the submitted papers and with the attendance at the workshop. There were both familiar and new faces at the workshop, and both familiar and new topics were being addressed.

A special issue of the "Journal of Object Technology" (JOT) dedicated to FTfJP 2003, with invited papers from the workshop, is planned.

7 List of Participants

Name	Email
Erika Ábrahám	eab@informatik.uni-freiburg.de
Davide Ancona	davide@disi.unige.it
Massimo Bartoletti	bartolet@di.unipi.it
John Boyland	boyland@cs.uwm.edu
Gilad Bracha	gilad.bracha@sun.com
Cees-Bart Breunesse	ceesb@cs.kun.nl
Patrice Chalin	chalin@cs.concordia.ca
Dave Clarke	dave@cs.uu.nl
Alessandro Coglio	coglio@kestrel.edu
Sophia Drossopoulou	sd@doc.ic.ac.uk
Manuel Fähndrich	maf@microsoft.com
Steve Freund	freund@cs.williams.edu
Allen Goldberg	goldberg@email.arc.nasa.gov
Johannes Henkel	henkel@cs.colorado.edu
Stephan Herrmann	stephan@cs.tu-berlin.de
James Hook	james.hook@cse.ogi.edu
Atsushi Igarashi	igarashi@kuis.kyoto-u.ac.jp
Bart Jacobs	bart.jacobs@cs.kuleuven.ac.be
Giovanni Lagorio	lagorio@disi.unige.it
Gary T. Leavens	leavens@cs.iastate.edu
K. Rustan M. Leino	leino@microsoft.com
Michael Moeller	Michael.Moeller@informatik.uni-oldenburg.de
Peter Müller	peter.mueller@inf.ethz.ch
David Naumann	naumann@cs.stevens-tech.edu
Richard Paige	paige@cs.york.ac.uk
Arnd Poetzsch-Heffter	poetzsch@informatik.uni-kl.de
Erik Poll	erikpoll@cs.kun.nl
Wishnu Prasetya	wishnu@cs.uu.nl
Michael Prasse	michael.prasse@tomcat.de
Christoph von Praun	praun@inf.ethz.ch
Axel Rauschmayer	axel@rauschma.de
Norbert Schirmer	norbert.schirmer@web.de
Igor Siveroni	siveroni@doc.ic.ac.uk
Matthew Smith	mjs198@doc.ic.ac.uk
Don Syme	dsyme@microsoft.com
Mirko Viroli	mviroli@deis.unibo.it
Tobias Wrigstad	tobias@dsv.su.se
Alexandre Zamulin	zam@iis.nsk.su

References

1. Luca Cardelli. Program fragments, linking, and modularization. In *Conference Record of POPL 97: The 24TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Paris, France*, pages 266–277–, New York, NY, January 1997. ACM, ACM.
2. S. Drossopoulou, S. Eisenbach, B. Jacobs, G. T. Leavens, P. Müller, and A. Poetzsch-Heffter. Formal techniques for Java programs. In Jacques Malenfant, Sabine Moisan, and Ana Moreira, editors, *Object-Oriented Technology. ECOOP 2000 Workshop Reader*, volume 1964 of *Lecture Notes in Computer Science*, pages 41–54. Springer-Verlag, 2000.
3. S. Eisenbach, G. T. Leavens, P. Müller, A. Poetzsch-Heffter, and E. Poll, editors. *Formal Techniques for Java-like Programs*. Technical Report 408, ETH Zurich, 2003. Available from `www.inf.ethz.ch/research/publications/show.php?what=408`.
4. B. Jacobs, G. T. Leavens, P. Müller, and A. Poetzsch-Heffter. Formal techniques for Java programs. In A. Moreira and D. Demeyer, editors, *Object-Oriented Technology. ECOOP'99 Workshop Reader*, volume 1743 of *Lecture Notes in Computer Science*, pages 97 – 115. Springer-Verlag, 1999.
5. G. T. Leavens, S. Drossopoulou, S. Eisenbach, A. Poetzsch-Heffter, and E. Poll. Formal techniques for Java programs. In A. Frohner, editor, *Object-Oriented Technology. ECOOP 2001 Workshop Reader*, volume 2323 of *Lecture Notes in Computer Science*, pages 30–40. Springer-Verlag, 2001.
6. G. T. Leavens, S. Drossopoulou, S. Eisenbach, A. Poetzsch-Heffter, and E. Poll. Formal techniques for Java-like programs. In J. Hernandez and A. Moreira, editors, *Object-Oriented Technology. ECOOP 2002 Workshop Reader*, volume 2548 of *Lecture Notes in Computer Science*, pages 203–210. Springer-Verlag, 2002.

Object-Oriented Reengineering

Serge Demeyer[1], Stéphane Ducasse[2], Kim Mens[3], Adrian Trifu[4], Rajesh Vasa[5], and Filip Van Rysselberghe[1]

[1] Dept of Mathematics and Computer Science, University of Antwerp, Belgium
[2] Software Composition Group, University of Berne, Switzerland
[3] Département d'Ingénierie Informatique, Université catholique de Louvain, Belgium
[4] Programmstrukturen, FZI Forschungszentrum Informatik, Karlsruhe, Germany
[5] Dept of Information Technology, Swinburne University of Technology, Australia

1 Introduction

The ability to reengineer object-oriented legacy systems has become a vital matter in today's software industry. Early adopters of the object-oriented programming paradigm are now facing the problems of transforming their object-oriented "legacy" systems into full-fledged frameworks.

To address this issue, a series of workshops have been organized to set up a forum for exchanging experiences, discussing solutions, and exploring new ideas. Typically, these workshops are organized as satellite events for major software engineering conferences, such as ECOOP'97 [5], ESEC/FSE'97 [10,11], ECOOP'98 [16,30], ECOOP'99 [14,13], ESEC/FSE'99 [12]. The last of this series so far has been organized in conjunction with ECOOP'03, its proceedings were published as a technical report from the University of Antwerp [8], and this report summarizes the key discussions and outcome of the workshop[1].

For the workshop itself we chose a format which balanced presentation of position papers against time for discussion, using the morning for presentation of position papers and the afternoon for discussion in working groups. Due to time restrictions we could not allow for every author to present. Instead, we invited three authors to not only present their own work, but also to summarize two related position papers. This format resulted in quite vivid discussions during the presentations, because authors felt more involved and because the three persons presenting (Roland Bertuli, Ragnhild Van Der Straeten, and Adrian Trifu) did such a splendid job in identifying key points in the papers. Various participants reported that it was illuminating to hear other researchers present their work.

During the discussion we maintained a list of "points of interest" on the blackboard, which later served as a guidance for identifying common issues. Based on this list, we broke up in two working groups, one on *Visualisation of Software Evolution*, the other on *Reengineering Patterns*. The workshop itself was concluded with a plenary session where the results of the two working groups

[1] The workshop was sponsored by the European Science Foundation as a Research Network "Research Links to Explore and Advance Software Evolution (RELEASE)" and the Fund for Scientific Research – Flanders (Belgium) as a Research Network "Foundations of Software Evolution".

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 72–85, 2004.

were ventilated in the larger group. Finally, we discussed some practical issues, the most important one being the idea to organize a similar workshop next year.

2 Summary of Position Papers

In preparation of the workshop, we received 12 promising position papers (none of them has been rejected) which naturally fitted into three categories: (a) Dynamic Analysis, (b) Design Consistency, (c) Methods and techniques in Support of Object-Oriented Software Evolution. After a reviewing phase, the authors were allowed to revise their initial submission and the resulting position papers were collected in the WOOR'03 proceedings [8]. These proceedings were sent out to all participants beforehand in order to allow them to prepare for the workshop.

For each of the categories, we asked one author to summarize the position papers; you will find these summaries below.

2.1 Dynamic Analysis

Understanding how a program under maintenance is structured, is an important step in re-engineering that application. Such program comprehension techniques either use static, i.e. source code and other documents, or dynamic, i.e. runtime information like method invocations, information. However the different nature of object-oriented languages with its late binding and polymorphism, makes it harder to rely solely on static information. This makes run-time analysis an important research topic within the field of object-oriented re-engineering.

The four papers on this subject of run-time analysis focussed on two different problems. Coping with the huge amount of information generated by run-time traces, was the concern of the first two papers ([31] and [4]). Where the third [21] and fourth [22] paper presented their solution to allow an optimal instrumentation of the code.

Using a Variant of Sliding Window to Reduce Event Trace Data. In [31], Zaidman and Demeyer present a technique based on the ideas of sliding window and frequency spectrum analysis. Using the execution frequency of methods, they partition the methods of a program in 3 groups: (1) those that are executed frequently (those methods point to low-level functionality, (2) the midrange and (3) those that are executed infrequently (very high-level, e.g. the main() of en program). According to the authors the main interest is in the midrange sector because these methods can give clues about a programs architecture. A variant of the sliding-window mechanism, well-known in the world of telecommunications, is used to pass over the execution trace, identifying regions containing a high degree of methods catalogued in the midrange category.

Run-time Information for Understanding Object-Oriented Systems. In [4], Bertuli, Ducasse and Lanza measure aspects of a running software system, such as the number of created instances or the number of method invocations. these measurements are then visualized using so-called 'polymetric views' [7,20]. The paper identifies four configuration of views and metrics that offer important information about the running system and the role of identified classes.

A Mechanism for Instrumentation Based on the Reflection Principle. / A New Strategy for Selecting Locations of Instrumentation. In [21], Li and Chen separate the instrumentation code from the actual code by using meta-level objects. of course, this approach has been shown to work for other languages such as Smalltalk and Java. Yet, these authors show that it is feasible to apply this approach for C++ systems as well. Since C++ lacks meta-objects, the authors actually rely on an *open compiler* to mix the meta-level code with the base objects.

Li and Chen submitted a second paper [22], reporting on a possible application of their open compiler approach for instrumenting code. The idea is to identify good places for instrumenting code by analyzing the call-graph. The approach is validated in a tool called XDRE.

2.2 Design Consistency

Four position papers addressed the problem of *design consistency*, i.e. the recurring problem of ensuring that the design documentation remains synchronized with all other project artifacts (i.e. requirements specifications, other design documents, the implementation). In principal, there are two different approaches to tackle design inconsistencies. The first one is the *horizontal approach*, attacking inconsistencies between different design documents; the other one is the *vertical approach*, tackling inconsistencies between models at different levels of abstraction. As a representative for the vertical approach, we had the paper [23], [17]. As a representative for the horizontal approach we had the papers [25], [19].

Intentional Source-Code Views. Mens and Poll [23] propose the lightweight abstraction of intentional source-code views as a way to codify high-level information about the architecture, design and implementation of a software system, that an engineer may need to better understand and maintain the system. They report on some experiments that investigate the usefulness of intentional source-code views in a variety of software maintenance, evolution and reengineering tasks, and present the results of these experiments in a pattern-like style.[2]

Maintaining Consistency Among UML Models. Mens, Van Der Straeten and Simmonds [25] address the problem of preserving consistency among the

[2] An extended version of this paper has been published at the ICSM2003 conference [24].

various UML models of which a software design typically consists, in particular after some of the models have evolved. To achieve the detection and resolution of consistency conflicts, the use of description logics and their associated tools is proposed. The authors argue how this approach allows them to partially automate the detection and resolution of design inconsistencies, thus increasing the maintainability of the software.[3]

Extending UML for Enabling Refactoring. In their position paper, Van Gorp et al [17] address the gap between existing UML tools on the one hand, and refactoring tools on the other. Whereas the former are designed to produce analysis and design models, the latter are designed to manipulate program code. Current tool vendors are trying to bridge this gap by regenerating program code from evolving UML models and vice versa (Model Driven Architecture is a typical example of this kind of approach). Including support for program code refactorings into the infrastructure of these novel UML tools is not trivial however. The authors describe some of the problems and propose a solution which they also implemented in a running tool prototype. [4]

Tracing OCL Constraints on Evolving UML Diagrams. Whereas the formal foundations of refinement and refactoring of program code have been widely studied, much less attention has been paid to formal methods for specification redesign and requirements tracing. A specification is usually spread through several documents and changes frequently during software development. As such it is very hard to trace the requirements and to validate compliance of a software system to its requirements. In his position paper, Kosiuczenko [19] studies the problem of tracing requirements in UML class diagrams with OCL constraints. He proposes a term rewriting approach to automatically derive traces which allow one to navigate through several specifications having different levels of abstraction as well as to trace requirements in forward and backward direction in distributed and changing specifications.

2.3 Methods and Techniques in Support of OO Software Evolution

A software system evolves as a consequence of the alternating phases of reengineering and functionality extensions. The process of reengineering [6,2] is typically composed of three main phases: a reverse engineering phase, a restructuring phase and a forward engineering phase, also referred to as change propagation.

Of the four submitted position papers that address the topic of object–oriented evolution, [1] and [29] deal with the reverse engineering and restructuring phases of reengineering respectively, [3] deals with aspects of software

[3] An extended version of this paper has been published at the UML2003 conference [28].

[4] An extended version of this paper has been published at the UML2003 conference [18].

metrics formalization, and [27] presents a method for investigating the evolution process of concrete systems, as a whole. A short summary of each paper is given below.

A Class Understanding Technique Based on Concept Analysis. In [1], Arévalo presents a novel method called *X-ray views*, which allows gaining insight into how a class operates internally, as well as how it interacts with other classes. This is useful when trying to understand a class in the context of reverse engineering a software system.

Understanding how a class works translates into identifying and evaluating different types of dependencies between the entities of the class: its instance variables, representing the state, and its methods, representing the behavior. By evaluating these dependencies and grouping them together, we can obtain a high level view of several aspects of the analyzed class, such as: (a) how clusters of methods interact to form together a precise behavior of the class; (b) how instance variables are related (i.e. used together); (c) what is the public interface of the class; (d) which methods are the so called "entry points" (methods that are part of the interface and that call other methods inside the class); (e) which methods use all of the class' state and which ones use only parts of it.

Dependencies are defined as sets of directly or indirectly related class entities, and are determined using Concept Analysis. For the unambiguous specification of dependencies, the author introduces a simple formalism, and using this formalism, defines a number of seven different dependencies. Some examples include *direct accessors*, *exclusive direct accessors*, *collaborating instance variables* and *interface methods*.

Based on the concept of dependencies, the author defines the higher level concept of *view*, as a combination of a set of dependencies. She then exemplifies the technique by defining two views *core attributes* and *public interface*. For example, the view called *public interface* is defined in terms of the two dependencies *interface methods* and *externally used state*. The defined views are applied on a concrete Smalltalk class, and results are discussed.

As future research, the author intends to define and evaluate more views, as well as investigate inheritance relations from the standpoint of the approach.

Strategy Based Restructuring of Object–Oriented Systems. In [29], Trifu and Dragoş present a method for object–oriented design improvement. The motivation behind their work is the conceptual gap that exists between state of the art design flaw detection techniques and current source code transformation technology. More precisely, the problem addressed is how to start from a given design problem, and (automatically) derive a sequence of source code transformations that eliminate the problem, while at the same time improve the system with regard to a predefined set of quality criteria. By design flaw, the authors refer to structural flaws in the code, caused by violations or misuse of principles and heuristics of good design (e.g. *god class*, *feature envy*, *data class*, etc. see [26]).

At the heart of the approach lies the novel concept of *correction strategy*, defined as a "structured description of the mapping between a specified design flaw and all possible ways to correct it". Selecting between alternative paths through a strategy is supported by the quality related information contained in the strategy itself, as well as a suitable quality model. The quality model, as specified by the methodology, is a prediction tool, used to estimate the quality impact of choosing one path from the several possible ones, thus enabling quality-aware decisions to be taken either automatically or manually.

Based on *correction strategies*, a quality-driven, highly automatable methodology for design improvement is presented. The methodology has the advantage that it allows for a great deal of flexibility by allowing the software engineer to configure the desired level of automation. This way, a tool that implements the methodology is able to function in a large number of configurations, ranging from a fully automatic to a fully assisted mode, in which the software engineer has complete control over all decisions taken in the restructuring process.

A set of correction strategies, together with the methodology and the quality model provides the missing link, and therefore a complete solution to the problem of design flaw removal from object–oriented systems.

The paper provides a critical overview of the state of the art in the field of restructuring software systems in general and design improvement in particular. As future work, the authors intend to continually improve on all aspects of the approach, provide a comprehensive catalogue of correction strategies, as well as perform a thorough evaluation of the approach on real-life case studies.

A Formal Library for Aiding Metrics Extraction. The approach presented in [3] does not directly address any of the phases of software evolution, but instead can be considered as a fundamental contribution to all reengineering approaches that are based on software measurement. In the paper, Baroni and Brito e Abreu present a library called FLAME (a Formal Library for Aiding Metrics Extraction), that can be used to formalize object–oriented design metrics definitions.

The declared purpose of the library is to encourage the use of software metrics among software designers by providing them with formalized design metrics, that are based on the UML model rather than source code. This way, the benefits of software measurements with increased tool support are made possible in early phases of software development, before source code is even available. Moreover, such definitions are at the same time formal and executable, thus making them appropriate for experiments replication (a recurring problem in the area).

The library contains a number of about 90 formally defined functions, at different levels of abstraction, such as *classifier*, *package*, *attribute* and *operation*. As formal language, the OCL (Object Constraint Language) is used on the core UML meta–model. FLAME functions are further classified as general, set, percentage and counting functions. Examples of FLAME functions include *new features*, *all attributes* and *defined operations*, defined at classifier context, and

classes number, *internal base classes* and *supplier classes*, defined at package context.

In the end, the authors present formal definitions for a few well known metrics from the literature and draw conclusions.

As future work, the authors intend to extend the scope of the library to the complete UML model, especially the behavioral parts. The authors also consider formalizing the same metrics but using different meta–models. As the goal of their research, the authors propose to provide precise definitions for most accepted sets of metrics, as well as the creation of a metrics framework which will allow the creation and comparison of metrics.

Method for Investigating the Evolution of Concrete Software Systems. Although it is a widely accepted fact that any software system must continue to evolve in order to remain successful, little is known today about how successful systems have evolved. Therefore little has been learnt from past experience and mistakes. This is the motivation behind the technique described by Van Rysselberghe and Demeyer in [27].

The approach presented in the paper tries to shed some light upon the nature of the phenomenon of evolution, by applying already established methods for software visualization and detection of duplication in large amounts of data, to successive versions of real-world systems. The lessons learnt in this way should improve our methods and understanding of software evolution.

The idea behind the approach is the same as the one used to reconstruct the past evolution processes of early life on earth: comparing successive releases of the software systems (the fossil remainders) and analyzing the differences. Differences in the implementation of two consecutive versions are highlighted using the technique presented in [15], where a two dimensional matrix, called a *dot plot diagram*, shows duplicate lines of code as dots and mismatched lines as empty space. A dot plot diagram corresponding to the comparison between a system and itself would be a square, and would only show a diagonal (a perfect diagonal means that every line of code is identical to itself). However, when comparing different versions of the same system, the original diagonal is broken up into segments that are shifted or cut out. By studying these changes and identifying patterns that correspond to known refactorings, one can reconstruct the evolution process of the system.

The authors exemplify the technique on the refactoring "pull-up-method", which is a typical refactoring used for eliminating duplicated code. It moves duplicated methods higher up the class hierarchy and replaces them by a single method to be reused by all subclasses. The corresponding patterns of change in the dot plot diagram are presented and analyzed in detail.

In order to increase the scalability of the approach, the authors propose the use of various data-reduction as well as automatic pattern recognition techniques.

3 Visualization of Software Evolution

The goal of the working group on *evolution visualization* was to discus different approaches to visualize the evolution of a software system. To achieve this goal two visualization techniques were presented and discussed by the participants. Although no general conclusions on how evolving programs should be visualized in order to understand their evolution were formulated, it did help both ideas presented to mature. Through the discussion possible improvements or applications were presented.

The focus of the first discussion was mainly on the visualization of object oriented software systems. Rajesh Vasa presented an initial idea of using hierarchical layering of classes in a given software system based on dependency analysis. Though details of the algorithm have not been worked out, the key concept is the use of Fan-In and Fan-Out metrics to layer the classes in a system. Once this hierarchical layering has been completed, one could see a visual representation of the software system where each layer had a number of classes. A number of these layer diagrams would have to be generated, one for each version of the software system under observation. Using animating algorithms we can then visualize the software system as it is changing. In the open discussion various participants put forward suggestions and ideas on how to get a better picture of these evolutions. For fine granularity one could focus on how a single class in the overall system and observe its evolution. Observing a single class or a set of classes can be very useful if the development team wanted to get an overview of how a set of classes evolved in the past. This type of observation would be very useful in determining quickly the set of classes that changed most (comparatively). Further, this approach can help identifying classes where the dependencies are changing frequently. Visual representation can be easily achieved as a simple chart where only the classes reaching a certain pre-defined threshold would be shown. The alternative approach discussed focused on how the overall layer diagram would change as the system evolved. This type of visualization will require a simple animation and provide a quick feedback of how the system was developed. The animation can provide an indication of the development methodology used, as in top-down or bottom-up. In a scenario where a development team built the library classes first and then focused on building the user-interface layers, we should see a bottom-up build up of the classes in the hierarchical layer diagram animation. This animation should mirror the development methodology used. The discussion did not identify specific uses for this, but it can be valuable for managers as it can provide insight into how the engineers are building the system and to ensure that the planned development approach is being undertaken.

The second discussion was focussed on the visualization of changes. Van Rysselberghe shortly presented the idea for using a combination of clone detection tools and dotplots to visualize the changes made from one version of a program to another [27]. For the visualization of changes, such technique is indeed a good solution certainly because it allows to spot changes rapidly. However as some noticed, some problems go together with this visualization. A first problem is

that of order. Reordering the contents of a source file, causes many mismatches which aren't interesting for its evolution. However a solution found for this problem is to use a pretty printer to format and order the attributes, methods, etc within a file. A harder problem is that of the visualizations scalability. Both the author as some of the participants didn't expect any help from filtering techniques which would be applied. However making the whole process a two step process is probably solving this problem. The idea is to use metrics first, just to find the possible changes which might have happened. The author himself, argued that using a some kind of similarity metric between files might succeed in solving the problem. After locating a file and its evolved counterpart, it would compare both using the prescribed technique, increasing the technique's and visualisation's scalability (applicability).

4 Reengineering Patterns

The goal of this workgroup was to 'mine' for new reengineering patterns, similar but complementary to those that can be found in the Reengineering Patterns' book [9]. The approach taken was to come up with a list of known reengineering problems first, and distill potential patterns out of each of these[5].

Each reengineering pattern description contains a specific *problem* related to software reengineering, an *example* of the problem, a proposed *solution* to that problem as well as some *trade-offs* regarding that particular solution. Given the limited time we had for discussion, the patterns we 'identified' should only be considered as 'potential' patterns. For example, we did not discuss the other parts of which a reengineering pattern consists: a *rationale* motivating the importance of the pattern, *known uses* of the pattern and *what next* to do after having applied the pattern. All reengineering patterns we identified are listed below.

4.1 Changing Libraries

Problem. A software application uses a certain library (or component) which we want to replace by a new one.

Example 1. We have a Java application that works with an Oracle database and want to modify it to work with a Sybase (or some other) database.

Example 2. We have an application of which the user interface is based on AWT and want to change it so that it uses SWING.

Solution. Rather then just modifying the existing application to make it work with the new library (or component), first add an intermediate layer in between the application and the library. Like this the application becomes much less coupled to the library it depends on. Given a sufficiently abstract layer in between, accommodating similar libraries in the future will become much easier.

[5] if the Reengineering Pattern book [9] did not already contain a pattern that addressed that particular problem

Trade-offs. The solution should not be used when writing a new intermediate layer would be too expensive. This is for example the case when the problem domain is not well-established yet, i.e. when it is difficult to find a possible abstraction for all libraries of a certain kind. For example, adding an intermediate layer that works with different database management systems is a quite standard and frequently occurring solution. Writing an intermediate layer that is independent of the particular user interface being used, on the other hand, is a task that should not be taken on lightly.

4.2 Predicting Change Impact

Problem. How can we predict the impact of changes to a software system?
Example. Changing, for example, the default value of a variable may have far reaching consequences throughout the entire implementation, wherever this value is explicitly or implicitly, directly or indirectly relied on.
Solution. Many tools and techniques exist to help in predicting the impact of changes to a program: program analysis, simulation on system model (dependency graph), testing after change is implemented. However, all these techniques have their limitations and there might always be cases (i.e. possible impacts) that we missed.
Trade-offs. Do not rely too much on the tools and techniques that exist. Sure, they will help, but be aware that they are not 'all powerful'. Also, the stronger the technique (i.e., the more exact it is) the less efficient it probably is.

4.3 Missed Abstractions

Problem. In 'bad' object-oriented programs, a lot of the problems are often due to a lack of abstraction, or having chosen the wrong abstractions.
Examples. Code duplication, wrong use of object-oriented concepts, overuse of global variables, long parameter lists, ...
Solution. Refactorings can often do a quite good job in rectifying some of these situations.
Trade-offs. Make sure that the 'bad code' is really 'bad code' and not optimized (or deliberately written in that style for some other important reason).

4.4 Eliminating Dead Code

Problem. How can we eliminate dead program code from an application?
Example. A method that is not called internally (from within the same class) neither from any other class. Why keep this method if it is not used anyway?
Solution. Although there are some tools and techniques that may help in detecting 'dead code', the problem is that one never really knows whether the code is really dead. Maybe it does not seem to be used in the application itself but it is called from other sources that you don't have; or maybe instead of being dead it is just 'not born yet' in the sense that someone already

put in place a piece of code with the idea of using it in some near future. Therefore, we suggest not to remove the dead code entirely but to replace it with assertions and keep the old code for a time (say one or two years or so), to play safe.

Trade-offs. It is difficult to decide how long to keep the code and who will take the final decision to remove the code in the end.

4.5 Enforcing Coding Conventions

Problem. How can we make sure that a given set of coding standards and conventions will be consistently used throughout an application?

Example 1. We want to ensure that the instance variables of a class are never called directly, but always through an accessing method.

Example 2. We want to ensure that all mutator methods of instance variables have a consistent name. For example, the naming convention adopted in Java is to name it after the name of the variable preceded by 'get'. The naming convention in Smalltalk is to name it after the name of the variable concatenated with a colon ':'.

Solution. Several research tools are available today that allow us to detect breaches of such naming conventions, as well as to enforce their consistent usage.

Trade-offs. However, care should be taken with enforcing these conventions if the development team is not entirely comfortable with them. For example, maybe the team already understands its code sufficiently well so that it is not really needed to 'clean it up' to make it more readable (by enforcing the use of certain conventions throughout the entire system).

4.6 Other Potential Reengineering Patterns

Some other reengineering problems we discussed but from which we did not have the time to extract a potential reengineering pattern are the following: (a) How to detect and solve usage of *data classes* in object-oriented programs. (b) How to detect and correct (run-time) *dangling references* in programs. (c) How to *ensure class cohesion* in object-oriented programs. (d) *Preserving behavioral contracts in inheritance relations*, i.e. making sure that inherited methods in subclasses respect the intended behavior of their parent methods. (e) *Controling external and internal quality specifications* of a system during reengineering.

5 Conclusion

In this report, we have listed the main ideas that were generated during the workshop on object-oriented reengineering. Based on a full day of fruitful work, we can make the following recommendations.

- *Viable Research Area.* Object-Oriented Reengineering remains an interesting research field with lots of problems to be solved and with plenty of possibilities to interact with other research communities (dynamic analysis, modeling, UML, metrics, visualization to name those that were touched upon during the workshop).
- *Establish a research Community.* All participants agreed that it would be wise to organize a similar workshop at next year's ECOOP.
- *Workshop Format.* The workshop format, where some authors were invited to summarize position papers of others worked especially well.

References

1. Gabriela Arevalo. X-ray views on a class using concept analysis. In Serge Demeyer, Stéphane Ducasse, and Kim Mens, editors, *WOOR'03 Proceedings*, 2003.
2. Holger Bär, Markus Bauer, Oliver Ciupke, Serge Demeyer, Stéphane Ducasse, Michele Lanza, Radu Marinescu, Robb Nebbe, Oscar Nierstrasz, Michael Przybilski, Tamar Richner, Matthias Rieger, Claudio Riva, Anne-Marie Sassen, Benedikt Schulz, Patrick Steyaert, Sander Tichelaar, and Joachim Weisbrod. The FAMOOS object–oriented reengineering handbook, 1999.
3. Aline Lucia Baroni and Fernando Brito e Abreu. A formal library for aiding metrics extraction. In Serge Demeyer, Stéphane Ducasse, and Kim Mens, editors, *WOOR'03 Proceedings*, pages 62–70, 2003.
4. Roland Bertuli, Stephane Ducasse, and Michele Lanza. Run-time information for understanding object-oriented systems. In Serge Demeyer, Stéphane Ducasse, and Kim Mens, editors, *WOOR'03 Proceedings*, pages 10–20, 2003.
5. Eduardo Casais, Ari Jaasksi, and Thomas Lindner. FAMOOS workshop on object-oriented software evolut ion and re-engineering. In Jan Bosch and Stuart Mitchell, editors, *Object-Oriented Technology (ECOOP'97 Workshop Reader)*, volume 1357 of *Lecture Notes in Computer Science*, pages 256–288. Springer-Verlag, December 1997.
6. Elliot J. Chikofsky and James H. Cross. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7(1):13–17, January 1990.
7. Serge Demeyer, Stéphane Ducasse, and Michele Lanza. A hybrid reverse engineering platform combining metrics and program visualization. In Francoise Balmas, Mike Blaha, and Spencer Rugaber, editors, *Proceedings WCRE '99 (6th Working Conference on Reverse Engineering)*. IEEE, October 1999.
8. Serge Demeyer, Stéphane Ducasse, and Kim Mens, editors. *Proceedings of the ECOOP'03 Workshop on Object-Oriented Re-engineering (WOOR'03)*, Technical Report. University of Antwerp - Department of Mathematics and Computer Science, June 2003.
9. Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. *Obje ct-Oriented Reengineering Patterns.* Morgan Kaufmann, 2003.
10. Serge Demeyer and Harald Gall, editors. *Proceedings of the ESEC/FSE Workshop on Object-Oriented Re-engineering*, TUV-1841-97-10. Technical University of Vienna - Information Systems Institute - Distributed Systems Group, September 1997.
11. Serge Demeyer and Harald Gall. Report: Workshop on object-oriented reengineering (WOOR'97). *ACM SIGSOFT Software Engineering Notes*, 23(1):28–29, January 1998.

12. Serge Demeyer and Harald Gall, editors. *Proceedings of the ESEC/FSE'99 Workshop on Object-Oriented Re-engineering (WOOR'99)*, TUV-1841-99-13. Technical University of Vienna - Information Systems Institute - Distributed Systems Group, September 1999.
13. Stéphane Ducasse and Oliver Ciupke. Experiences in object-oriented re-engineering. In Ana Moreira and Serge Demeyer, editors, *Object-Oriented Technology (ECOOP'99 Workshop Reader)*, volume 1743 of *Lecture Notes in Computer Science*, pages 164–183. Springer-Verlag, December 1999.
14. Stéphane Ducasse and Oliver Ciupke, editors. *Proceedings of the ECOOP Workshop on Experiences in Object-Oriented Re-engineering*, FZI report 2-6-6/99. FZI Forschungszentrum Informatik, June 1999.
15. Stéphane Ducasse, Matthias Rieger, and Serge Demeyer. A language independent approach for detecting duplicated code. In Hongji Yang and Lee White, editors, *Proceedings ICSM'99 (International Conference on Software Maintenance)*, pages 109–118. IEEE, September 1999.
16. Stéphane Ducasse and Joachim Weisbrod, editors. *Proceedings of the ECOOP Workshop on Experiences in Object-Oriented Re-engineering*, FZI report 6/7/98. FZI Forschungszentrum Informatik, July 1998.
17. Pieter Van Gorp, Hans Stenten, Tom Mens, and Serge Demeyer. Enabling and using the uml for model driven refactoring. In Serge Demeyer, Stéphane Ducasse, and Kim Mens, editors, *WOOR'03 Proceedings*, pages 37–40, 2003.
18. Pieter Van Gorp, Hans Stenten, Tom Mens, and Serge Demeyer. Towards automating source-consistent uml refactorings. In *Proc. 6th International Conference on the Unified Modeling Language*. Springer Verlag, 2003.
19. Piotr Kosiuczenko. Tracing requirements during redesign of uml class diagrams. In Serge Demeyer, Stéphane Ducasse, and Kim Mens, editors, *WOOR'03 Proceedings*, pages 41–47, 2003.
20. Michele Lanza and Stéphane Ducasse. Polymetric views — a lightweight visual approach to reverse engineering. *IEEE Transactions on Software Engineering*, 29(9):782–795, sep 2003.
21. Qingshan Li and Ping Chen. A mechanism for instrumentation based on reflection principle. In Serge Demeyer, Stéphane Ducasse, and Kim Mens, editors, *WOOR'03 Proceedings*, pages 21–25, 2003.
22. Qingshan Li and Ping Chen. A new strategy for selecting locations of instrumentation. In Serge Demeyer, Stéphane Ducasse, and Kim Mens, editors, *WOOR'03 Proceedings*, pages 26–31, 2003.
23. Kim Mens and Bernard Poll. Supporting software maintenance and reengineering with intentional source-code views. In Serge Demeyer, Stéphane Ducasse, and Kim Mens, editors, *WOOR'03 Proceedings*, pages 32–36, 2003.
24. Kim Mens, Bernard Poll, and Sebastián González. Using intentional source-code views to aid software maintenance. In *Proceedings of ICSM2003*, 2003.
25. Tom Mens, Ragnhild Van Der Straeten and Jocelyn Simmonds. Maintaining Consistency between UML Models with Description Logic Tools. In Serge Demeyer, Stéphane Ducasse, and Kim Mens, editors, *WOOR'03 Proceedings*, 2003.
26. Arthur J. Riel. *Object–Oriented Design Heuristics*. Addison–Wesley, first edition, 1996.
27. Filip Van Rysselberghe and Serge Demeyer. Studying software evolution using clone detection. In Serge Demeyer, Stéphane Ducasse, and Kim Mens, editors, *WOOR'03 Proceedings*, pages 71–75, 2003.

28. Ragnhild Van Der Straeten, Tom Mens, Jocelyn Simmonds, and Viviane Jonckers. Using description logic to maintain consistency between UML models. In *Proc. 6th International Conference on the Unified Modeling Language*. Springer Verlag, 2003.
29. Adrian Trifu and Iulian Dragos. Strategy based elimination of design flaws in object-oriented systems. In Serge Demeyer, Stéphane Ducasse, and Kim Mens, editors, *WOOR'03 Proceedings*, pages 55–61, 2003.
30. Stéphane D ucasse and Joachim Weisbrod. Experiences in object-oriented reengineering. In Serge Demeyer and Jan Bosch, editors, *Object-Oriented Technology (ECOOP'98 Workshop Reader)*, volume 1543 of *Lecture Notes in Computer Science*, pages 72–98. Springer-Verlag, December 1998.
31. Andy Zaidman and Serge Demeyer. Using a variant of sliding window to reduce event trace data. pages 4–9.

Mobile Object Systems: Resource-Aware Computation

Ciarán Bryce[1] and Crzegorz Czajkowski[2]

[1] Object Systems Group,
University of Geneva, Switzerland
Ciaran.Bryce@cui.unige.ch
[2] Sun Microsystem Laboratories,
Mountain View, California, USA
Grzegorz.Czajkowski@sun.com

1 Introduction

The ECOOP Workshop on Mobile Object Systems is now in its 9th year. Over the years, the workshop has dealt with topics related to the movement of code and data between application platforms, security, operating system support, application quality of service, and programming language paradigms. In many cases, the workshop has been a forum to discussed traditional object-oriented issues, since mobility influences such a broad spectrum of topics.

The theme of this year's workshop was resource-aware computing. As the object-oriented and mobility paradigms are increasingly applied to multi-user, cluster and application server environments, the ability to measure and control resource consumption is crucial. Related issues include resource accountability, resource trading, performance, and security. The workshop combined with last year's ECOOP Workshop on Resource Control.

The object-oriented and mobility paradigms are being used in environments where resources, e.g., files, CPU, memory, network, servers, etc. are shared between several users. In these environments, it is important to ensure a fair distribution of resources among users. This issue will become even more important as object-oriented environments are developed for clusters and application servers. Resource awareness is important since an unbalanced distribution of resources can lead to poor application performance. It is also important for security reasons, since a program's ability to monopolize a resource can lead to a denial of service attack.

Despite its importance, resource control is one of the least understood issues in object-oriented programming today. It is for this reason that we saw the need for a workshop on the issue. We invited a high-quality program committee from both industry and academia, many of whom were present at the workshop.

The workshop took place on Monday, July 21st, in Darmstadt University, just prior to the main ECOOP conference. The format of the workshop was informal. It consisted of presentations and a series of discussions on these. The topics treated included resource management, but also issues relating to mobility. The number of attendees varied, reaching 15 at times.

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 86–91, 2004.

2 The Talks

The first talk was an invited talk from **Niranjan Suri** of Florida University, USA. The talk was entitled *Using Mobility to Achieve Agent Agility*. The scenario for this includes *ad hoc* resource sharing and future combat systems. The environments are ones where location, network connectivity and latency as well as device availability vary a lot over time. The aim is therefore to opportunistically discover and exploit available resources to improve reliability, efficiency and survivability. Niranjan Suri presented several agile scenarios and open research issues.

Grzegorz Czajkowski of Sun Microsystems then presented *A Resource Management Interface for the Java(TM) Platform*. This is work conducted with Stephen Hahn, Glenn Skinner, Pete Soper (Sun Microsystems, Inc.) and Ciaran Bryce (University of Geneva).

An area where safe language platforms seriously lag behind operating systems is that of resource management (RM) facilities. Preventing denial of service attacks, providing load balancing, and monitoring the usage of a given resource are all difficult to do at all and impossible to do well. The talk presented a resource management interface (RM API) for the Java platform with the following features:

- The interface is applicable to a variety of resource management scenarios.
- Flexibility. The interface enables managing a broad range of resource types.
- Extensibility. The resource management interface supports the addition of new resources in a uniform manner.
- The interface hides from applications whether a given resource is managed by the underlying operating system (OS), by the Java Virtual Machine (JVM(TM)), by a core library, or by trusted middleware code.
- The interface does not require an implementation to depend on specialized support from an OS or from hardware.
- The interface is applicable to all kinds of Java application environments, from the J2ME(TM) platform to the J2EE(TM) platform.

This proposal allows an expressive set of resource management policies to be coded. A computation can be dynamically bound to such a policy. Programs can reserve resources in advance and thus ensure predictable execution. Applications can install resource monitoring code so that pro-active programs can observe resource availability and take any actions required to ensure performance and availability or to ward off denial of service attacks. Existing applications can still run without modification, even if the Java Development Kit (JDK(TM)) classes or Java runtime environment (JRE) they depend on exploits the resource management framework. The interface does not impact in any way on how actual resource managers (schedulers, automatic memory subsystems, etc.) should be written - these managers become RM-enabled by inserting calls to RM API methods to convey information about resource consumption attempts. The effort required to retrofit existing resource managers is thus minimized.

The authors have prototyped the interface entirely in the Java programming language. Modifying the virtual machine for that purpose can be more efficient, although the performance achieved with the current prototype is satisfactory for practical use. To a large extent, this is due to the fact that the interface allows for managing tradeoffs between the precision of resource accounting and its cost.

More detail is given in a technical report at
`http://research.sun.com/techrep/2003/abstract-124.html`.

Michal Cierniak from Intel Corporation then presented *Resource Use in the Interaction of Managed and Unmanaged Code*, work conducted with Marsha Eng, Neal Glew, Brian Lewis, and James Stichnoth.

Mobile and resource aware systems are often implemented using managed runtime environments (MREs) because most of the code is managed by a Virtual Machine (VM). The VM can keep track of and control the use of resources, which provides a good basis for resource control mechanisms. However, some code is not managed, in particular, the native code underlying native methods.

The authors discussed the transition from managed to native code in their MRE implementation. In particular, they discussed two issues in carrying out these transitions: implementing all the details for the transfer, and passing reference parameters. Also discussed were two approaches to each issue having different performance and resource requirements. First, the authors presented the implementation of the transition from managed to native code using customized stubs versus using a generic stub interpreter, and showed that there is a memory versus performance tradeoff for these two approaches. Second, they described two ways of implementing JNI object handles and showed that a slightly more complex approach is much better than a simple one in terms of both memory usage and performance. These demonstrate the importance of choosing the right mechanisms for managed to native transitions in both resource limited and high performance virtual machines.

The fundamental tradeoff between the customized stubs and stub interpreters for invoking JNI methods is that the customized stubs are more efficient since they do minimum amount of work during each transition by specializing the stub once at compile-time. On the other hand, the stub interpreters are much easier to implement, debug and modify. Better memory use is a secondary benefit of stub interpreters since per method code is very small. In the experiments, the authors compared performance and memory usage for SPECjvm98 and SPECjbb benchmarks. Customized stubs always provide better or equal performance: the difference can be as high as a factor of 8. Stub interpreters excel in memory usage: they use an order of magnitude less memory than custom stubs. In their VM, the authors chose to use customized stubs because performance is very important for their research and they can accept higher memory usage and the extra engineering effort to develop and maintain the compiler for custom stubs.

The next talk was presented by **Kari Schougaard** from the Center for Pervasive Computing in Aarhus University, Denmark. The co-author of the work is Ulrik P. Schultz.

The focus on mobile devices is continuously increasing, and improved device connectivity enables the construction of pervasive computing systems composed of heterogeneous collections of devices. the authors see application mobility as a central concern in such an environment; whatever activity a user is performing can be supported by applications that can migrate to devices in close physical proximity to the user.

The position paper presented at the workshop describes an ongoing effort to implement a language and a virtual machine for applications that execute in a pervasive computing environment. This system, named POM (Pervasive Object Model), supports applications split into coarse-grained, strongly mobile units that communicate using method invocations through proxies. The authors are currently investigating efficient execution of mobile applications, scalability to suit the availability of local resources (primarily for interaction with the user), and language design issues.

In POM, a mobile entity is referred to as a system (in the sense of a self-contained entity). Systems are normally employed in POM to modularize applications. A single virtual machine can host multiple systems that communicate using proxies. A system can at any time migrate to some other virtual machine where it continues its execution (e.g., strong migration of all threads). It is thus a set of objects that migrate as a single unit.

POM applications are normally structured into several systems that communicate using proxy calls. By isolating uses of native resources (e.g., access to GUI or to files) in a single system (which will be linked to the location because of the use of native resources), the remaining application can freely migrate to other virtual machines while maintaining an indirect reference to the native resources of the original virtual machine (in the form of proxies). Alternatively, the application can decide to obtain access to the resources of the virtual machine that it has migrated to, for example by replacing a remote GUI with a local one.

In the talk at the workshop, the use of systems with strong mobility was motivated by a scenario at a hospital: a nurse equipped with a PDA is distributing medicine. Mobile systems can migrate from the PDA to server machines to retrieve information from an electronic patient record. Also, systems can migrate onto machines with larger displays to display and interact with the retrieved information. Structuring the objects of the running program into systems facilitates determining which objects should be moved to another device in an environment with given resources in the form of various devices.

Andrew Wils from Leuven University presented *Component contracts for local resource awareness*. The co-authors of the contribution were Joris Gorinsek, Stefan Van Baelen, Yolande Berbers, and Karel De Vlaminck.

Ubiquitous and pervasive computing paradigms tend to shift towards dynamically reconfigurable service-oriented platforms. Embedded devices will host

services that are required to reconfigure, adapt and update (in short, to evolve). Multimedia services will run side by side with other, less resource-hungry, yet perhaps more important services. On the other hand resource availability on ubiquitous hardware may itself change, e.g. a device may get less power or bandwidth. It is clear to see that the stability of the whole platform can be jeopardized if it is not aware of the varying resource needs and availability. A global distribution of services among hardware devices is insufficient to solve these problems.

A resource allocation must be found and maintained to satisfy all the needs of a particular configuration of services on a device. The system software on a device needs to be aware of the minimum requirements of all services, and the easiest way to obtain this information is to ask the services. Software needs to be aware - in an abstract way - of the changing limitations of the underlying system. They propose a component-contract based approach to solve these challenges and a local resource-aware system that takes into account the individual QoS levels each service can function in. Software components interact with this system to agree on a resource contract. The resource-aware system needs to know, predict or limit up-front how a component is going to be used, if it is to guarantee correct behavior. Therefore a component declares its resource costs and its timing constraints for deployment and intended use. Resource costs may depend on other components, whose details may not be known until they are deployed. To cope with this, resource contracts capture the dependencies between components. Component dependencies are resolved at runtime to make a feasibility analysis of the intended use of the component. The outcome of this will determine the QoS level of a component and form an agreement between the component itself and the components it uses.

Resource contracts have been applied in a prototype component runtime system, in which a broker ensures the contract negotiation and re-negotiation with components and a monitor keeps components from violating their resource contracts.

Carlos Varela from Rensselear Polytechnic Institute in New York presented a talk entitled *Mobility and Security in Worldwide Computing*. The co-author of the talk was Robin Tolle. The talk looked at program component mobility and security as fundamental stepping stones towards flexible yet robust distributed computing systems. Mobility and security introduce new requirements on software, e.g., it is critical to devise strategies for secure and controlled distributed resource management. The authors specified an actor-based model and formalism to reason about program component mobility and secure resource access in worldwide computing systems. Specifically, they introduce a simple actor language as an extension to the lambda calculus with its operational semantics, and extend the language with mobility, location-awareness, and access control primitives. The Secure Mobile Actor Language (SMAL) forms a basis for reasoning about mobility and security in distributed systems, and serves as a specification for practical language implementations.

3 Conclusions

The consensus at the end of the workshop was that the workshop series should continue, even if the original theme of the workshop – mobile object systems – is actually less and less present. The workshop is becoming a forum for highly focussed, sometimes speculative, emerging and downright difficult issues.

The striking feature of the resource awareness issue is that it covers a wide spectrum. The workshop saw a presentation of an API for resource management in Java environments, safe implementation techniques for virtual machines, and policy (contracts). It seems as if resource management touches on every issue of computer science. Despite this diversity, the attendees felt that there is a real benefit in accommodating this diversity under the umbrella of a single workshop.

We should lastly mention that all papers and presentations of this year's workshop and of all previous editions are available at our website:
`http://cui.unige.ch/~ecoopws`

Quantitative Approaches in Object-Oriented Software Engineering

Fernando Brito e Abreu[1], Mario Piattini[2], Geert Poels[3], and Houari A. Sahraoui[4]

[1] QUASAR Research Group, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Monte da Caparica, Portugal
[2] ALARCOS Research Group, Departamento de Informática, Universidad de Castilla-La Mancha, Ciudad Real, Spain
[3] Department of Management Information, Operations Management and Technology Policy, Faculty of Economics and Business Administration, Ghent University, Ghent, Belgium and Centre for Industrial Management, Katholieke Universiteit Leuven, Leuven, Belgium
[4] Département d'Informatique et Recherche Opérationnelle, Université de Montréal, Montréal, Quebec, Canada

Abstract. The QAOOSE'2003 workshop brought together, for a full day, researchers and practitioners working on several aspects related to quantitative evaluation of software artifacts developed with the object-oriented paradigm. Ideas and experiences where shared and discussed. This report includes a summary of the technical presentations and subsequent discussions raised by them. Eleven out of twelve submitted position papers were presented, covering different aspects such as metrics formalization, new metrics (for coupling, cohesion, constraints or dynamic behavior) or metrics validation, to name a few. In the closing session the participants were able to discuss open issues and challenges arising from researching in this area, as well as they tried to forecast which will be the hot topics for research in the short to medium term.

1 Historical Background and Motivation

Throughout several editions[1], the QAOOSE series of workshops has attracted participants, mostly from academia and research centers somehow related to industry, which were involved or interested in the application of quantitative methods in object-oriented software engineering research and practice. Quantitative approaches in the realm of the object-oriented paradigm is a broad but active research area that aims at the development and evaluation of numerical methods, techniques, tools and practical guidelines to assess, forecast and improve the quality of software products and the efficiency and effectiveness of software processes.

2 Workshop Overview

Since this a quantitative workshop, let us start with some numbers. Seventeen (17) people, out of thirty-three (33) overall participants (if we include the co-authors that were not present) attended the workshop. They were representing eighteen (18)

[1] Information on those previous editions can be found in the end of this report.

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 92–106, 2004.

different organizations from ten (10) different countries. Among the attendants, five (5) people were not authors, as it is normally the case in these kind of closed[2] workshops. They have asked the organizers to attend the workshop, which is an additional evidence of the interest raised by this area, as well as an additional measure of success of this year's edition.
This workshop encompassed four 90-minute sessions, each chaired by one of the four organizers, with the exception of Mario Piattini that asked to be replaced by Coral Calero, from his own research team. The topics of each session were, respectively: (1) Quantitative approaches using UML, (2) Components and Run-time, (3) Inter and intra class connectivity evaluation and (4) Estimation. On each session three presentations took place, except for the last session where, unfortunately, none of the authors of a position paper originating from the Indian Institute of Technology (Ram, Reddy and Rajasree) did show up. Each presentation, plus corresponding discussion, took around 30 minutes. Those presentations were based on submitted position papers, that went through a revision process conducted by each of the corresponding session chairs. In the final 30-minute wrap-up session, a collective effort was performed to draw some conclusions and to identify future trends for this workshop. Due to the number of participants, active participation
In the next four sections we will present the abstract of each of the position papers presented in each of the workshop sessions, adapted from the one submitted by the corresponding authors. Next, a view of each paper contribution, produced by an independent evaluator (the session chair) is put forward. Finally, a summary of the comments arisen from the consequent discussion among the workshop participants, collected by each session chair, is included. After each position paper title, the list of its co-authors is indicated and the presenter is identified. A table with detailed information about each workshop participant is included in the end of this report.

3 Official Web Site

The call for papers, plus the full text of all accepted position papers are available at: http://ctp.di.fct.unl.pt/QUASAR/QAOOSE2003/

4 Session 1 – UML and Product Lines (Chair: Houari Sahraoui)

An OCL-Based Formalization of the MOOSE Metric Suite
Aline Lúcia Baroni, Fernando Brito e Abreu (presenter)

Authors abstract: Most design metrics proposed in the literature are ill-defined, mainly due to its informality. This leads to misinterpretation, namely while producing tools to automate their collection and hampers widespread comparison of obtained metrics values, required to achieve an understanding of their evaluative and predictive

[2] Participation in ECOOP workshops is usually guaranteed through the submission of a position paper.

power. Besides, since those informal metrics are often not defined upon a design metamodel, their application becomes inadequate in the realm of current industry standard practices for software design. So, in spite of a growing number of empirical research studies, design metrics usage is still not a widespread practice in the software industry.
The well-known MOOSE metrics set, from Chidamber and Kemerer is formalized in this paper, showing, among other things, how the formalization process can help to identify metrics that are unsuitable at design-time. The adopted formal approach to metrics definition was proposed earlier by the QUASAR research group, and uses OCL expressions upon the UML metamodel. Besides solving the above-mentioned ambiguity problems, this approach allows using the metrics definition directly in automating the metrics collection process.

Independent view: The goal of this position paper is to point out the problem of interpretation of existing metrics. Indeed, for most of the existing object-oriented metrics, their definitions are ambiguous and their collection will depend on the target programming language. To circumvent this problem, the authors propose an approach for formalizing object-oriented metrics. This approach is based on the constraint language OCL. It was implemented as a library (FLAME) and tested on a set of metrics (MOOSE metrics).
In its current version, the approach is defined and applied only on structural design metrics (those extracted from UML class diagrams). The authors are currently investigating the possibility of extending it to consider metrics that deal with the behavioral aspects.

Comments summary: Many aspects of this proposal were discussed. First, some participants were interested by the applicability of the approach on dynamic and behavioral diagrams of UML and in particular with the version 2.0. The authors were aware about this aspect and said that is an ongoing work.
The second aspect concerned the usefulness of the approach on the particular case of reverse engineering and language semantics. The authors comment that the main goal of the approach is to work on UML diagrams (forward engineering) and in the case of reverse engineering (metric collection from source code) some problems remain unsolved.

Measuring OCL Expressions: a "Tracing"-Based Approach
Luis Reynoso, Marcela Genero, Mario Piattini
Presented by Coral Calero (non-author)

Authors abstract: Since class diagrams constitute the backbone of object-oriented (OO) software development, many metrics were proposed to measure internal quality attributes such as the structural complexity, coupling, size, etc. However, none of the proposed metrics take into account the added complexity when class diagrams are complemented by Object Constraint Language (OCL) expressions. OCL expressions improve class diagrams enhancing their semantic properties, adding precision, improving their documentation and comprehensibility. The importance of OCL and the lack of defined metrics for OCL expressions motivated us to propose a set of

metrics for the structural properties of OCL expressions. The first set of metrics proposed by the authors considers only those OCL concepts related to a "tracing" technique. The authors believe that the "tracing" technique affects the cognitive complexity and, by consequence, the understandability of OCL expressions and the maintenance of a UML class diagram. The main goal of this paper is to show the definition of a set of metrics for structural properties of OCL expressions in a methodological way. The theoretical validation of these metrics according to a property-based framework proposed by Briand et. al is also presented.

Independent view: The objective of this position paper is to discuss the importance of measuring OCL expressions as part of UML design, which is not done yet in the literature and practice. They propose a set of metrics for structural properties of OCL expressions. The authors describe a method that can be used for defining and validating formally and empirically the metrics proposed. Those metrics consider the OCL concepts related to a "tracing" technique which, in the authors' view, affects the cognitive complexity, and by consequence the understandability of OCL expressions and the maintenance of a UML class diagram. The proposed metrics are then theoretically validated using a property-based approach.

Comments summary: The focus of the discussion was on the described method rather than the metrics for OCL themselves. The role of some tasks of this framework (empirical validation, acceptation, etc.) were explained and motivated by the authors.

Quality Modelling for Software Product Lines

Adam Trendowicz (presenter), Teade Punter

Authors abstract. In today's embedded software systems development, non-functional requirements (e.g., dependability, maintainability) become more and more important. Simultaneously the increasing pressure for developing software in shorter time and at a lower cost pushes software industry towards product lines solutions. To support product lines for high quality embedded software, quality models are required. In this paper it is investigated to which extent existing quality modeling approaches facilitate high quality software product lines. First, several requirements for an appropriate quality model are defined. Then, those requirements are used to review the existing quality modeling approaches. The authors conclude from the review that no single quality model fulfils all of the identified requirements. However, several approaches contain valuable characteristics. Based upon those characteristics, the Prometheus approach is proposed. Prometheus is a goal-oriented method that integrates qualitative and qualitative approaches to quality control. The method starts quality modeling early in the software lifecycle and is reusable across product lines.

Independent view: In this position paper, the authors consider the problem of defining quality models for product line solutions. They discuss the possible extents to the existing quality modeling approaches that can facilitate the production of high quality software product lines. To this end, they define the requirements that a quality model must satisfy and use those requirements to review the existing quality modeling approaches. The results of this review show that no single quality model

fulfills all of the requirements. To solve this problem, the authors propose a goal-oriented approach called Prometheus.

Comments summary: One of the characteristics that quality model should have for product lines is the integration of the context in the evaluation. This point was particularly discussed during the workshop. The authors commented that the usage of Bayesian belief networks (BBN) can help considering this characteristic. However, due to confidentiality reasons, it was not possible to see a concrete example of this.
Another part of the discussion was dedicated to the problem of the definition of the BBN parameters (conditional probabilities). The authors consider that domain experts should do this. Some participants argued that this is not obvious and an interesting direction to investigate is automatic learning.

5 Session 2 – Components and Run-Time (Chair: Geert Poels)

A Survey on the Quality Information Provided by Software Component Vendors
Manuel F. Bertoa, José M. Troya, Antonio Vallecillo (presenter)

Authors abstract. The last decade marked the first real attempt to turn software development into engineering through the concepts of Component-Based Software Development (CBSD) and Commercial Off-The-Shelf (COTS) components. The idea is to create high-quality parts and join them together to form a working system. One of the most critical processes in CBSD is the selection of the COTS components from a repository that meet the user requirements.
Current approaches try to propose appropriate quality models for the effective assessment of such COTS components. These proposals define quality characteristics, attributes and metrics, which are specific to the particular nature of COTS components and CBSD. However, the authors have found that the information required to evaluate components using those quality models and metrics is not available in the existing commercial software repositories. In this paper a survey carried out on the most popular COTS component vendor sites is presented, trying to evaluate how much of the information required to assess COTS components is actually available. The authors' goal was to estimate the current gap between the "required" and the "provided" information, since there is no point in defining theoretical measures for COTS components if the data they rest upon is not available. Analyzing this gap is the first step towards successfully bridging it, by both refining the component quality models so their metrics are more realistic, and by improving the information currently provided by software component vendors.

Independent view: In this paper, the authors analyze the gap between the information provided by component vendors and the information that is required to evaluate component quality as part of the COTS component selection process in component-based software development (CBSD). A sample of 164 components from ComponentSource, today's major component marketplace, was used to examine the quality information that is typically offered by component vendors through their web-based systems. The information found was used to assess which of the attributes of

the author's previously developed COTS Component Quality Model (COTS-QM), presented in the QAOOSE'02 workshop, could be measured.
The results show that, based on the information provided, less that half of the attributes in COTS-QM can be used to evaluate quality. In particular, there is almost no information for the quality attributes that have to be measured at run-time. Moreover, the information found was mostly in a raw form (e.g. help-files, manuals, demos, UML diagrams) and required further processing before it could be used in the quality model. Based on their study, the authors recommend that component vendors and acquirers (and quality theoreticians) agree on the type and form (e.g. tagged values like "<metric, value>") of the quality information that is provided. In this context the authors argue for the development of simple, but realistic quality models. They also recommend independent quality assessments and the use of web services for the retrieval of the requested quality information.

Comments summary: The discussion of the paper focused on the quality information that should be minimally available at the vendor's site and on the chances of success regarding the agreement that must be reached with the component vendors. The authors comment that it is still too early to decide on the quality information that must be provided as the proposed quality metrics for components are not validated yet. They do not expect an agreement with vendors in the short term. However, analogous to what happened in the hardware industry, with a push from the buyers, it can be done.

Toward a Definition of Run-Time Object-Oriented Metrics
Aine Mitchell (presenter), James F. Power

Authors abstract. This position paper outlines a programme of research based on the quantification of run-time elements of Java programs. In particular, the authors adapt two common object-oriented metrics, coupling and cohesion, so that they can be applied at run-time. They demonstrate some preliminary results of the analysis they performed on programs from the SPEC JVM98 benchmark suite.

Independent view: Building on the object-oriented metrics definition work of Chidamber and Kemerer, the authors of this paper propose two new coupling metrics (Dynamic CBO for a class, Degree of Dynamic Coupling) and two new cohesion metrics (Simple Dynamic LCOM, Dynamic Call-Weighted LCOM) to measure the external, respectively internal complexity of object-oriented software at run-time. They argue that these metrics deal with the behavioral aspects of a software system, measuring program complexity as actually observed during program execution, whereas the extensive research in the area of object-oriented metrics has focused almost exclusively on metrics for evaluating the structural aspects of a system. As such, these static metrics only quantify what may happen if a program is executed. Several examples were given where the coupling and cohesion of the software at run-time was different from what has been predicted based on a static analysis of the software code. Especially in cases of state-based behavior, dynamic binding, and unequal frequencies of method invocation or instance variables access, run-time metrics provide information of software complexity, not obtained through the classic

(static) metrics. The authors demonstrated their claims by analyzing the complexity of programs from two benchmark suites for the Java programming language. They showed graphically that the run-time profiles of these programs exhibit different levels of coupling and cohesion than the values resulting from calculating the Chidamber and Kemerer CBO and LCOM metrics.

Comments summary: Most of the paper discussion related to the possible use of the information provided by the proposed run-time metrics. Apart from the detection of certain anomalies in the code and possible implications for quantifying the effectiveness of software testing strategies, the authors plan to investigate by means of correlation studies the impact of run-time complexity on external quality attributes. It was also observed that caution must be exercised when interpreting the results of a dynamic analysis because of the dependency on the scenarios and input data that are used. Apart from defining run-time metrics, the research of the authors is therefore aimed at defining a methodology for the sound application of the metrics. They also plan to use their metrics on other than benchmark Java programs, as real-world applications will certainly show different behavior.

CQM: A Software Component Metric Classification Model
Joaquina Martín-Albo (presenter), Manuel F. Bertoa, Coral Calero, Antonio Vallecillo, Alejandra Cechich, Mario Piattini

Authors abstract. In the last few years component-based software development (CBSD) has been imposed as a new paradigm in software systems construction. CBSD is based on the idea that software systems can be constructed by selecting and integrating appropriate components, which have already been developed, and then assembling them to obtain the functionality desired in the final application. Multiple authors have proposed metrics to quantify several components characteristics in order to help in its selection. Nevertheless, rather than helping developers, such proposals often provoke more confusion due to the fact that they do not systematically take into account different aspects of the components.
Trying to achieve clarity in this line, the authors have developed the CQM model (Component Quality Model), whose first aim is to propose a classification of the defined metrics for software components. The model will also be suited both for the evaluation of a component or a component system. Finally, it is necessary to indicate that this is the first version of the model, and that it requires refinement by means of its use and discussion in different fora.

Independent view: This paper reports upon a joint effort of researchers from several universities into the quality of component-based software development (CBSD). The authors present the Component Quality Model (CQM), a comprehensive quality evaluation framework for components and component-based systems. The CQM model proposes four dimensions that must be considered in the quality evaluation: (i) the quality characteristic (distinguishing between internal quality, external quality and quality in use properties), (ii) the granularity (system or component) and visibility (internal or external) dimension, (iii) the lifecycle process involved and (iv) the stakeholder involved. Due to its comprehensive nature, the model integrates, in some

of its dimensions, previously proposed quality models for components, like the COTS-QM model of Bertoa and Vallecillo, thereby linking with the first 'quality in CBSD'-related paper in this session.
A first use of the model is to classify the metrics that have been proposed in the literature for the characterization, evaluation and selection of components. The authors demonstrate this idea by classifying a number of usability metrics along the four dimensions. They also point out that the model presented is only a first version, which needs to be further refined. The new version of the model does, for instance, not consider the stakeholder, as this dimension does not seem to be orthogonal to the life cycle process.

Comments summary: During the paper discussion the intended use of the CQM model was clarified. It was suggested that the metrics classification could be used to identify both neglected spaces (and hence initiate new metric definition work) and conflicts, like when researchers propose the same metrics, but with different names. It was further stressed that the ultimate goal of CQM is to evaluate components and component-based systems, giving stakeholders an instrument to select the appropriate quality metrics. Regarding a workshop participant's question on the validity of classified metrics in particular contexts, it was answered that for the usability metrics classified, their validity was not really questioned. On the other hand, the authors point out that besides the classification along the four dimensions, other metric characteristics need to be identified, including their theoretical and empirical validation, their objective or subjective nature, and whether there is automated support for their collection and calculation.

6 Session 3 – Inter and intra Class Connectivity Evaluation (Chair: Fernando Brito e Abreu)

A New Class Cohesion Criterion: An Empirical Study on Several Systems

Linda Badri, Mourad Badri (presenter)

Authors abstract: Class cohesion is considered one important object-oriented software attribute. Cohesion refers to the degree of the relatedness of the members in a class. High cohesion is a desirable property of classes. Several metrics have been proposed in the literature in order to measure class cohesion in object-oriented systems. They capture class cohesion in terms of connections among members within a class. The major existing class cohesion metrics are essentially based on instance variables usage criteria. It is a special and a restricted way of capturing class cohesion. Most of these metrics have been experimented and widely discussed in the literature. They do not take into account some characteristics of classes as stated in many papers. The authors believe that class cohesion metrics should not exclusively be based on common instance variables usage. They present, in this position paper, a new criterion, which focuses on interactions between class methods and propose a new class cohesion metric. The authors have developed a cohesion measurement tool for Java programs and performed a case study on several systems. The selected test

systems vary in size and domain. The obtained results demonstrate that their new class cohesion metric captures several pairs of related methods, which are not captured by the existing cohesion metrics.

Independent view: The authors argue that the traditional view of class cohesiveness, based upon measuring the usage of attributes by operations, is just a partial view of that property. Therefore, they explore another dimension of class cohesiveness, based upon the interaction among class operations or, in other words, dependent on the internal invocation of operations. They call this property the functional cohesion. The authors then proposed two metrics for this property, DC_D which is the percentage of public operation pairs that are directly related and DC_I which is the percentage of public operation pairs that are indirectly related. That relatedness is evaluated using only attribute cohesiveness or attribute and operation cohesiveness together.
An empirical study using a cohesion measurement tool for Java programs is also reported. That tool, built by the authors on top of a well-known parser (antlr), allows computing several published cohesion metrics, plus the ones defined in this paper. A data set of six free downloadable Java systems, summing up more than 2000 classes, was used. The experiment concluded that there is statistically significant difference between the cohesiveness metrics calculated with attribute and operation together, and the ones that consider attributes usage only.

Comments summary: A participant commented that it is not unexpected that adding a factor to an existing cohesion metric, would naturally increase the number of connected class pairs. Another participant confirmed with the presenting author that the proposed metrics are only static, that is, obtained from the design structure and not from running hypothetical scenarios. It was commented that the proposed metrics could also be applied to components, in a white box approach. Finally, the need for empirical validation of the proposed metrics usefulness was arisen.

Towards a Validation Process for Measuring Coupling: Integrating Axiomatic and Empirical Approaches

Miguel Lopez, Valérie Paulus (presenter), Naji Habra

Authors abstract: The validity of the measures used in software engineering is a critical matter about which no consensus has yet emerged, although it has prompted hard discussion. There is a need for unambiguous definitions of the mathematical properties that characterize the major measurement concepts. Such a mathematical framework could help to generate consensus among the software engineering community. The goal of this paper is to provide a formal validation process for software measurement. It presents a global measurement framework that integrates theoretical and empirical validation processes based on measurement theory.
The concept underlying the framework is to formalize some properties of the measure to be analyzed, and then to verify the conformity of these properties to the measure by means of formal experimentation.
This validation process determines a contextual validity (scope) defined by the set of factors or validity conditions that impact the validity of the measure. The paper

develops a case study that, under specified conditions, validates the Coupling Between Objects (CBO) as a measure of coupling.

Independent view: This paper deals with the recurring issue of validating software measures. In particular, it aims at presenting a process that combines theoretical (measurement theory based) and empirical validation. To illustrate the process, a case study is presented, that tries to validate the C&K Coupling Between Object Classes metric as a coupling metric. The authors use UML class diagrams to review the different coupling types among classes. They restrict their coupling scope to attribute and operation dependency.
The kind of validity the authors aim to deal with has to do with suitability, rather than usefulness. For instance, given a metric, they want to validate its suitability as a coupling metric, rather than validating it as a useful predictor or indicator of a given external characteristic, such as maintainability. The proposed validation process is composed of eight steps. It involves a set of OO design experts, responsible for (i) defining a required set of axioms mathematically and (ii) for sorting a set of fragments of class diagrams supposed to illustrate, as exhaustively as possible, alternative coupling topologies (called coupling paradigms in the paper). The former requirement is left unexplained since the authors use a set of well-known published axioms.
The two last steps of the proposed process are the ones where the validation is actually performed, being the first of them related to the empirical world and the used measurement instrument, and the second about the preservation of the relationship between the empirical and the numerical worlds.
Although the authors raise doubts on the problem of how to gain confidence on experts judgement and also on the representativeness of the coupling paradigms, the result of their validation case study allows to invalidate the Coupling Between Object Classes metric as a coupling metric.

Comments summary: The two authors present were questioned on the non-appropriateness of the Spearman correlation coefficient for order scales. One participant also mentioned the need for cross validation of the case study results. Another participant also questioned the "blind" adoption of the Briand and Morasca axioms and mentioned, for instance, how the axiom of non-negativity could be undesirable. The discussion motivated by this paper ended around the always-pervasive problem of obtaining representative sets for validation purposes.

Evolution of Cyclomatic Complexity in Object Oriented Software
Rajesh Vasa (presenter), Jean-Guy Schneider

Authors abstract: It is a generally accepted fact that software systems are constructed and gradually refined over a period of time. During this time, code is written and modified until stable releases of the system emerge. Many researchers have studied systems over a longer period of time in order to understand how they change and evolve. Despite these efforts, the authors feel that there is still a lack of a precise understanding how various properties of software change over time, in particular in the area of object-oriented systems. Such an understanding is of great importance if one wants to come up with techniques to provide feedback on the

evolution of quality and predictions about further evolution of software systems. Historically, collection of sufficient data to build useful models was not practical as source code and build histories were not freely available. The authors argue that by focusing their attention towards Open source software repositories, they will have a better hope building predictive models to help developers and managers. In this paper, the authors report on an exploratory study analyzing Open source object oriented software projects and present a first predictive model based on this analysis.

Independent view: The authors claim that few works were published on the evolution of OO systems, despite the availability of evolution data stored in open source repositories. The paper includes data from 5 Java projects throughout time (data ranging from 7 to 10 different versions each). Using a public domain metrics collection tool (JavaNCSS), the authors analyzed and commented the evolution of NCSLOC (non-commented source lines of code), confirming Lehman's "first law of software evolution". Then they analyzed the cyclomatic complexity of operations for which an implementation was available. The operations were grouped in five categories ranked with cyclomatic complexity intervals. Observation of results led to the confirmation of Lehman's "fifth law of software evolution" for object-oriented software.
The authors look after producing a predictive model of cyclomatic complexity based on the observed data, that could warn producers of any abnormal evolution. However, that model is left for future work in this paper.

Comments summary: A few questions emerged after the paper presentation. One was on the distribution of size versus complexity. The co-author present claimed that one had no impact on the other. He also mentioned that they are currently working on evaluating the "regularity" of graphs displaying the complexity categories throughout time. A participant suggested that the evolution of the shapes of the curves of cyclomatic complexity should be compared.
A question was also risen by one participant on the need for such a evolution study: if a large system evolves through non-disruptive steps, where consecutive versions differ very few, why bother measuring the difference? The author replied that the large number of plain selectors and modifiers contributed a lot to that regularity. If those operations were not considered, than the distinctions would be much more noticeable.

7 Session 4 – Estimation (Chair: Coral Calero)

Analogy-Based Software Quality Prediction
David Grosser, Houari A. Sahraoui (presenter), Petko Valtchev

Authors abstract: Predicting the stability of object-oriented systems is an important and challenging task. Classical approaches to quality prediction perform some form of inductive inference starting from datasets of software items with known quality factor values and looking for typical features that discriminate the items regarding the quality factor. However, most of the effective methods for predictive model

construction are based on the implicit hypothesis that the available samples are representative, which is rather strong. The approach the authors propose implements a similarity-based comparison principle. In it, the quality factor (stability) of a given software item is estimated from the recorded stability of a set of other items that have been recognized as the most similar to that item among a larger set of items stored in a database. This approach is evaluated using the successive versions of the JDK API.

Independent view: The authors propose an approach for predicting the stability of object-oriented systems. This approach implements a similarity-based comparison principle: the stability of a given software item is estimated from the recorded stability of a set of other items that have been recognized as the most similar to that item among a larger set of items stored in a database. The authors use CBR (Case-Based Reasoning) as a suitable approach to the software stability prediction problem because they have carried out some experiments that confirmed the usefulness of CBR for tasks where few theoretical knowledge is available. Finally, the authors think that this technique can also be applied to other quality factors prediction.

Comments summary: During the questions, authors were asked about the dependency of this estimation technique on the metrics selected. They argued that since this is the problem with estimation in general, they tried to apply the technique under the hypothesis of metrics set independence. Another question was why they prefer to use prediction instead of pair correlation. The presenter replied that pair-correlation is a good technique for knowing if a variable has an impact in another one, but not for estimation.

Definition and Validation of a COSMIC-FFP Functional Size Measure for Object-Oriented Systems

Geert Poels (presenter)

Author abstract: COSMIC Full Function Points is an ISO approved functional size measurement method for modeling and sizing software systems, based on their functional user requirements. Recently, a number of mappings have been proposed from the COSMIC-FFP meta-model onto the concepts used in UML and other OO modeling approaches. Given the many problems with COSMIC-FFP's main predecessor, Function Points Analysis, it is necessary to establish confidence in the validity of COSMIC-FFP as a functional size measure for OO systems, before its widespread diffusion in quantitative OO software engineering practice. In this paper the author presents an attempt at validating COSMIC-FFP using distance-based software measurement. This theoretical validation approach for software measures is firmly grounded in Measurement Theory and has been applied before in the validation of OO software measures.

Independent view: The author presented the formal validation of the COSMIC-FFP using the distance-based software measurement, an approach based on the measurement theory that has been previously applied to OO software measures. COSMIC Full Function Points is an ISO approved functional size measurement method for modeling and sizing software systems based on their functional user requirements.

The COSMIC-FFP metamodel of functional user requirements (FUR) for modeling and sizing a software system includes a set of modelling concepts. This metamodel is used to map the FUR that are captured in the operational requirements engineering or software engineering artifacts into a uniform COSMIC-FFP FUR model, which is subsequently measured. The COSMIC-FFP measurement function takes a piece of software as its argument and returns a value, representing the functional size of that piece of software. The mapping rules to generate a COSMIC-FFP FUR model are designed to be applicable to a broad range of artifacts, such as high-level software specifications that are documented in UML class diagrams and use case diagrams, UML, Real-time Object-Oriented Modelling methods, etc.
The final objective of the author was to prove if the metric could be considered valid with respect to the measurement theory. A theoretically validated measure means that we can evaluate the construct validity of the measure when used as a measurement instrument in an empirical study. The attempt to validate the COSMIC-FFP measurement function, as a ratio scale level functional size measure for OO software systems, was not successful. However, the distance-based software measurement demonstrates that the view of functional size that is inherent in the COSMIC-FFP metamodel allows measuring this property at the ratio scale level.

Comments summary: One workshop participant questioned the need to establish a mapping between the concepts in the COSMIC-FFP metamodel and the concepts used in UML and other OO modeling approaches. After all, COSMIC-FFP is designed to be independent of modeling paradigms. The author replied that although the COSMIC-FFP metamodel is generic and highly abstract, the application of COSMIC-FFP in practice requires more specific and detailed rules that must necessarily be tailored to the modeling paradigm used. Another question related to the measurement theoretic structures on which distance-based software measurement is based. It was answered that this approach to theoretical metrics validation belongs to an advanced type of measurement that is known as segmentally additive proximity measurement. The author presented more details about this approach at the QAOOSE'99 workshop.

8 Conclusions

In the final wrap-up session the name of the QAOOSE workshop series was questioned. There was a suggestion to broaden up the scope of the workshop to quantitative approaches for other than object-oriented modeling, specification and programming methodologies and technologies. In particular, workshop participants agreed that quality and process related research for component, web services and agent-based systems should fit into the workshop. To some extent, this year's edition has already paved the way to this, as it can be confirmed by reading the call for papers. It was also suggested not to change the name of the workshop series, since the underlying technology for these newer trends in software engineering is mostly still object orientation. Nevertheless, the organizers expressed the will to extensively review the list of workshop topics, namely by considering more recent development paradigms, thus opening the workshop participation to a broader audience.
This edition of the workshop was definitely one of the most successful ones, mainly due to an active participation of all attendees. The support and suggestions raised by

the new ideas presented by some participants and the feeling of interaction and interest of all the people in the workshop has resulted on prospects for collaborative research among different participants.
Let us finish this report with a strategy remark. Looking back to the participant lists of this edition, plus those of previous ones, we cannot avoid the observation that the vast majority of participants originated from academia and research centers, the latter somehow in-between universities and industry. In other words, QAOOSE has failed to attract people from "pure" industry, a situation that the QAOOSE organizers feel is undesirable, for bridging the gap between object oriented quantitative software engineering research and practice. This situation arises from the fact that this workshop is an embedded event of ECOOP, which, in turn, is an event dominated by and mainly targeted at academic researchers. Therefore, it may be a good idea to collocate QAOOSE with some other event where academic participation is not so much dominant.

9 Participants' Affiliation and Contacts

The following table includes detailed information about all participants in this workshop, with their corresponding roles (O - Organizer; A - Author or co-author; P - Present in workshop).

Name	Surname	Affiliation	Country	Email	Roles
Miguel	Lopez	CETIC Falcutés de Namur	Belgium	malm@cetic.be mlo@info.fundp.ac.be	A
Valérie	Paulus	CETIC Falcutés de Namur	Belgium	vp@cetic.be vpa@info.fundp.ac.be	AP
Naji	Habra	CETIC / Falcutés de Namur	Belgium	nha@info.fundp.ac.be	A
Adam	Trendowicz	Fraunhofer IESE	Germany	trend@iese.fhg.de	AP
Geert	Poels	Ghent University Katholieke Universiteit Leuven	Belgium	geert.poels@ugent.be geert.poels@econ.kuleuven.ac.be	OAP
P. Jithen-dra	Kumar Reddy	Indian Institute of Technology	India	jithendra@cs.iitm.ernet.in	A
M. S.	Rajasree	Indian Institute of Technology	India	rajasree@cs.iitm.ernet.in	A
D. Janaki	Ram	Indian Institute of Technology	India	djram@cs.iitm.ernet.in	A
James	F. Power	National University of Ireland	Ireland	jpower@cs.may.ie	A
Aine	Mitchel	National University of Ireland	Ireland	ainem@cs.may.ie	AP
Jean-Guy	Schneider	Swinburne Univers. of Technology	Australia	jschneider@swin.edu.au	A
Rajesh	Vasa	Swinburne Univers. of Technology	Australia	rvasa@swin.edu.au	AP
Linda	Badri	Univ. du Québec à Trois-Rivières	Canada	linda_badri@uqtr.ca	A
Mourad	Badri	Univ. du Québec à Trois-Rivières	Canada	mourad_badri@uqtr.ca	AP
Alejandra	Cechich	Univ. Nacional del Comahue	Argentina	acechich@uncoma.edu.ar	A

Luis	Reynoso	Univ. Nacional del Comahue	Argentina	lreynoso@uncoma.edu.ar	A
Coral	Calero	Universidad de Castilla-la-Mancha	Spain	coral.calero@uclm.es	AP
Marcela	Genero	Universidad de Castilla-la-Mancha	Spain	marcela.genero@uclm.es	A
Joaquina	Martín-Albo	Universidad de Castilla-la-Mancha	Spain	jmartin@proyectos.inf-cr.uclm.es	AP
Mario	Piattini	Universidad de Castilla-la-Mancha	Spain	mario.piattini@uclm.es	OA
Manuel	F. Bertoa	Universidad de Malaga	Spain	bertoa@lcc.uma.es	AP
José	M. Troya	Universidad de Malaga	Spain	troya@lcc.uma.es	A
Antonio	Vallecillo	Universidad de Malaga	Spain	av@lcc.uma.es	AP
Aline	Baroni	Universidade Nova de Lisboa	Portugal	baroni@di.fct.unl.pt	A
Fernando	Brito e Abreu	Universidade Nova de Lisboa INESC-ID	Portugal	fba@di.fct.unl.pt fba@inesc.pt	OAP
Hervé	Paulino	Universidade Nova de Lisboa	Portugal	herve@di.fct.unl.pt	P
David	Grosser	Université de Montréal	Canada	grosserd@iro.umontreal.ca	A
Houari	Sahraoui	Université de Montréal	Canada	sahraouh@iro.umontreal.ca	OAP
Petko	Valtchev	Université de Montréal	Canada	valtchev@iro.umontreal.ca	A
Hans	Stenten	University of Antwerp	Belgium	hans.stenten@ua.ac.be	P
Filip	Van Ryssel-berghe	University of Antwerp	Belgium	filip.vanrysselberghe@ua.ac.be	P
Soren	Gaardbo Jensen	University of Copenhagen	Denmark	gaardbo@dikv.dk	P
Katharina	Mehner	University of Paderborn	Germany	mehner@upd.de	P

10 Related Workshops

This workshop is a direct continuation of the QAOOSE series of workshops, held at previous ECOOP conferences:

Workshop Name	**Conference/ Place**	**Organizers**
QAOOSE'2002 http://alarcos.inf-cr.uclm.es/qaoose2002	ECOOP'2002 Malaga, Spain	Mario Piattini, F. Brito e Abreu, Geert Poels, Houari Sahraoui
QAOOSE'2001 http://www.iro.umontreal.ca/~sahraouh/qaoose01/	ECOOP'2001 Budapest, Hungary	F. Brito e Abreu, Brian Henderson-Sellers, Mario Piattini, Geert Poels, Houari Sahraoui
QAOOSE'2000 http://ecoop2000.unice.fr/Program/Technical/Workshops/w10.html	ECOOP'2000 Cannes, France	F. Brito e Abreu, Geert Poels, Houari Sahraoui, Horst Zuse
QAOOSE'99 http://ecoop99.di.fc.ul.pt/techprogramme/w20.html	ECOOP'99 Lisbon, Portugal	F. Brito e Abreu, Walcelio Melo, Houari A. Sahraoui, Horst Zuse
OO Product Metrics for Software Quality Assessment http://www.crim.ca/~hsahraou/oopm.html	ECOOP'98 Brussels, Belgium	Houari A. Sahraoui, Sandro Morasca, Walcelio Melo
Quantitative Methods for OO Systems Development http://ctp.di.fct.unl.pt/QUASAR/ECOOP95	ECOOP'95 Aarhus, Denmark	Horst Zuse, Brian Henderson-Sellers, F. Brito e Abreu

Composition Languages

Markus Lumpe[1], Jean-Guy Schneider[2], Bastiaan Schönhage[3], Markus Bauer[4], and Thomas Genssler[4]

[1] Iowa State University, USA
lumpe@cs.iastate.edu
[2] Swinburne University of Technology, Australia
jschneider@swin.edu.au
[3] Compuware Europe B.V., The Netherlands
Bastiaan.Schonhage@nl.compuware.com
[4] Forschungszentrum Informatik Karlsruhe, Germany
{Bauer,Genssler}@fzi.de

Abstract. This report gives an overview of the Third International Workshop on Composition Languages (WCL 2003). It explains the motivation for a third workshop on this topic and summarizes the presentations and discussions.

1 Introduction

Workshop Goals

A component-based software development approach mainly consists of two steps: (i) the specification and implementation of components and (ii) the composition of components into composites or applications. Currently, there is considerable experience in component technology and many resources are spent for the first step, which resulted in the definition of component models and components such as CORBA, COM, JavaBeans, and more recently Enterprise JavaBeans (EJB), the CORBA Component Model (CCM), and .NET. However, much less effort is spent in investigating appropriate techniques that allow application developers to express applications flexibly as compositions of components. Existing composition environments mainly focus on special application domains and offer at best rudimentary support for the integration of components that were built in a system other than the actual deployment environment.

The goal of this workshop was to continue the fruitful discussions that have been established in the two previous workshops on composition languages (WCL '01 and WCL '02). These events have prepared the ground for a common understanding of distinguishing properties of composition languages. The most important questions that have been discussed in these events were

- what do we actually want to compose,
- how do we compose, and
- why do we want to compose.

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 107–118, 2004.

By answering these questions, the workshop participants were able to agree upon the thesis that composition is the explicit description of connections between components by means of connectors. Both components and connectors can be represented in the same system, i.e., a specially trimmed composition language.

To foster an even better understanding of the particular nature of composition languages, in this workshop we also wanted to focus on representation strategies for architectural software assets, in particular as a paradigm shift from component-centric development to model-centric and architecture-centric development was observed. One of the recent developments in this area is the Model Driven Architecture (MDA) defined by the Object Management Group (OMG). MDA is considered to be the next step in solving software integration problems as it introduces a separation between application logic and infrastructure by encapsulating infrastructure specific aspects as far as possible in code generators. In this context, we wanted to address the following questions:

- What are benefits and limits of model-centric approaches?
- How can component behavior be specified on a conceptual level?
- How can an existing set of components be integrated with model-centric approaches?

This briefly summarizes the context in which the goals of this workshop were originally defined. More specifically, we wanted to emphasize important issues of (i) the design and implementation of higher-level languages for component-based software development, (ii) approaches that combine architectural description, component configuration, and component composition, (iii) paradigms for the specification of reusable software assets, (iv) expressing applications as compositions of software components, and (v) the derivation of working systems using composition languages and components.

Submission and Participation

In the call-for-submission, authors were encouraged to address any aspect of the design and implementation of composition languages in their position statements. Furthermore, we were particularly interested in submissions addressing formal aspects of the issues mentioned in the previous section as well as case studies of using composition languages for real-world applications. The following list of suggested topics was included in the original call-for-submissions:

- Model-centric and architecture centric approaches:
  - Support for the specification of software architectures and architectural assets,
  - Interoperability support,
  - Design and implementation strategies for cross-platform development,
  - Programming paradigms for software composition,
  - Model- and architecture-centric development and composition methods,
  - Using existing components in model-centric and architecture-centric approaches, model extraction,

  - Modeling of components, specifically component behavior,
  - Mapping of architectural models to applications, model transformations,
  - Benefits of model-centric and architecture-centric approaches,
  - Case studies and success stories of model-centric and architecture-centric development,
  - Tool support for model-centric and architecture-centric development.
- Compositional reasoning:
  - Representation strategies for functional and non-functional properties,
  - Prediction of properties of compositions from properties of the involved components,
  - Reasoning about correctness of compositions,
  - Specifying and checking of architectural guidelines.
- Aspect of Composition languages:
  - Higher-level abstractions for composition languages,
  - Implementation techniques for composition languages,
  - Scalability and extensibility of the language abstractions,
  - Analysis of runtime efficiency of compositional abstractions,
  - Formal semantics of composition languages,
  - Type systems for composition languages,
  - Domain-specific versus general composition languages,
  - Case studies of composition language design,
  - Case studies of system development using composition languages,
  - Tool support for composition languages,
  - Taxonomy of composition languages.

All workshop submissions were evaluated in a formal peer-reviewing process. Review was undertaken on full papers by at least two members of the paper selection committee and selection was based on their quality and originality. Eight papers were accepted for presentation at the workshop and publication in the subsequent proceedings.

2 Presentations

The morning of the workshop was dedicated for two presentation sessions with three presentations each (unfortunately the authors of two papers could not attend the workshop). The first session covered issues in relation to the formal semantics for composition languages, whereas the second session was dedicated to composition frameworks. This section briefly summarizes the main issues of the presentations given.

The first presentation of the workshop was given by Joseph Kiniry about semantic component composition. His work was motivated by the fact that various specification languages for software components exist, but there are still many open question in relation to (i) searching components, (ii) composing components, and (iii) reasoning about composition given component specifications in any of these specification languages. Using JML and EBON as examples, he introduced some of the problems of how the semantics of annotated components

are currently defined. Based on these observations, he motivated the introduction of semantic components, a unification of constructs found in many specification languages. He illustrated the benefits and applicability of semantic components and concluded his talk with a summary of the main contributions.

The second talk in this session was given by Jørgen Steensgaard-Madsen about a family of composition languages. He presented a number of examples to illustrate composition of both homogeneous and heterogeneous components for a Unix-platform. The notion of an *explicit continuation* was advocated as useful in composition languages with a parametric type system, because such a continuation can serve as recipient of 'tuples' that may have types as elements, i.e., values of dependent types. Operations with multiple explicit continuations were advocated as a simple class-like notion that generalizes structured statements. A tool to implement interpreters for the family of composition languages was characterized and related to Chomsky's hierarchy of languages. He further advocated that an appropriate separation of implementations into a common *deep*-structure and a special *surface*-structure makes it is possible to construct languages incrementally by separate contributions.

The last presentation of the first session was given by Wishnu Prasetya (a joint submission with T.E.J. Vos, S.D. Swierstra, and B. Widjaja). He presented a theory to compose distributed components, which is based on reasoning about temporary interference and non-interference. This specialized theory requires that components synchronize in a certain way. In exchange it gives more information to the designer as to how two components can be constrained so that they match. The additional compatibility constraints it generates are also relatively light weight to verify. He concluded his talk by giving an outline how the approach is currently being integrated into an existing composition environment.

The first presentation of the second session was given by Naiyana Tansalarak (a joint submission with Kajal T. Claypool) who presented *XCompose*, an XML-based component composition framework. This work was motivated by the problem that to increase the overall usability of components, approaches for component composition frameworks must support *flexibility*, *extensibility*, *re-usability*, *correctness*, and *portability* during the composition phase. The presented approach was based on the hypothesis that complex component compositions can be broken down into a sequence of a primitive composition operators glued together by a simple language and the paradigm "application = components + composition language" where "composition language = operators + glue logic." Based on this hypothesis, she introduced *XCompose* and illustrated its five essential ingredients: (i) primitive composition operators, (ii) composition patterns, (iii) composition templates, (iv) composition contracts, and (v) *XDescriptors*, descriptions of compositions using XML. She concluded the presentation with some discussions for future thought and development.

The second presentation in the session on composition frameworks was given by Arne Berre. In his talk, he presented results and experiences made in the context of the COMBINE project, a Europen Union supported project that investigated the applicability of OMG's Model Driven Architecture (MDA) for component-based development in an industrial context. In particular, he fo-

cused on how the MDA compares with traditional workflow architectures and how composition of workflows can be analyzed using UML activity diagrams. He concluded his presentation by presenting various workflow patterns and comparing flow standards using the patterns as comparison criteria.

The final talk of the workshop was given by Rainer Neumann (a joint submission with Andreas Heberle) on model-driven development of component-centric applications in the context of the CompoBench research project. Given the problem that business applications are not developed from scratch any more, the goal of the project was to combine both tool and component technologies in such a way that business analyst and designers can write executable specifications. In the presentation, he outlined a number of requirements that such an approach must meet, in particular a process-based composition model with well-understood terminology, as well as their MDA-based solution of an integrated toolset. He concluded his presentation with lessons learned from using an MDA-based approach and discussed a list of open questions.

3 Discussion Sessions

The afternoon of the workshop was dedicated to discussing various issues which were brought forward during the morning presentations. At the beginning of the afternoon, an initial list of discussion topics for the two discussion sessions was elaborated. This list contained the following items:

1. Support for Model-driven composition
2. Notations for expressing compositions (is XML really suitable?)
3. Unification/integration of approaches from various areas
4. Generic composition abstractions, composition patterns
5. Verification/correctness of compositions
6. Foundations to express semantics of compositions
7. Inter-language bridging in composition languages

It was then decided to split up the participants into two breakout groups. The first group agreed to discuss items 1 to 4 of the list whereas the second group was discussing items 4 to 7. In the following, we will briefly summarize the results of the two breakout groups.

3.1 Model-Driven Composition

The first breakout group started with working out a definition for "composition." All participants agreed with the following, informal definition: *composition means putting things together to form an ensemble that works together to achieve one goal.* For many group members, composition was seen as a bottom-up process to which a reverse, top-down process "decomposition" exists. The latter matches the human way of breaking up a complex problem into smaller chunks which can be dealt with independently.

In component based software engineering, the decomposition of a system into components is often influenced by

- reuse concerns (what is already there to be reused?) and
- the component model (technical infrastructures).

But what does composition really mean in practice? It was first noted that composition may involve composing entities (i.e., composition is done on a data centric view) or activities (i.e., composition involves combining operations). It was also noted that proper composition should combine components from the same abstraction level, e.g. components that represent concepts of the business domain (business components or objects).

The group then discussed the need of adapting components before composing them. The following observations could be established:

- Adaptation often is required to overcome mismatches/different assumptions between components that arise from technology or architecture issues.
- A homogeneous and standardized runtime environment can avoid many adaptation problems.
- In an ideal world, composition does not need to look inside the components, but it will rely on proper specifications that provide enough information to facilitate composition. In that case, adaptation should be deferred to some technical abstraction level and could be done with tool support or avoided altogether by infrastructure code generators, as they are employed by MDA tools.
- Still, there seem to be many cases where adaptation is required, e.g. for filling in missing functionality or for coping with different semantics of information (pricing information may be stored in multiple currencies).

After having discussed these basic terms of (de-)composition and adaptation, the group moved on to examine how the concepts of components and composition can be matched with the concepts of OMG's MDA initiative. It was agreed to discuss this issue in the context of business applications, i.e., applications that support the business processes of an enterprise.

The core concept of MDA is that software development is done model-driven. To build a system, a model is developed and then step-by-step refined into executable code. In most cases, this refinement is done with strong tool support. From the OMG's standards, three particular models seem to be relevant for composition:

- The *entity or data model* (UML class diagram) provides a data or entity centric view on components. It is the most basic model, since most business applications focus on modifying some (persistent) information captured in business objects.
- The *activity model* (UML activity diagram) is used to express the functional decomposition of a system into business functions.
- *Component diagram:* the mapping of these functions to components results in a component diagram which defines component boundaries, interfaces of components and interactions between components (via ports and connectors). A key issue here seems to be the specification of the data flow between

activities. Often the reverse way is needed, however: when reusing existing components, the functionality they provide has to be matched with the activity model.

In order to express composition of components in terms of these three models, enough information to derive lower abstractions from these models have to be provided. Furthermore, this information must be comprehensive enough to enable component composition at an implementation level.

Most members of the discussion group see a lack of logical links and consistency rules between the three models. To overcome this, special rules, patterns, and stereotypes have to be defined. According to some group members, this calls for a domain, company or project specific methodology when combining model-based approaches with components and composition. For example, such a methodology could state the rule that each (activity) component implements a single activity. Many participants, however, were not sure whether the forthcoming and now accepted UML 2.0 standard will be able to improve that situation.

Some people in the composition language community have been working on defining base operations for composition. An interesting observation could be made in context with the OMG models above: some base operations (like $m1; m2$ or $m1|m2$) deal with the composition of control-flows and therefore only affect the activity models, other operations (like the aggregation operation from the XCompose paper) primarily affect the entity model or the component diagram.

All participants of the first breakout group left the discussion session with the impression that it is necessary and worthwhile to spend some more research on how to combine concepts brought up by MDA people with concepts developed by the composition language community. This would, however, require that MDA experts become familiar with composition concepts and theory, and that composition experts study the latest material from the MDA and OMG initiatives.

3.2 Verification Support

The second breakout group started with the question "what is a component?" in order to have some common understanding about terminology and concepts. Not surprisingly, various definitions were brought forward, for example

- a component is a thing that can be composed,
- a component is a black-box abstraction with plugs, or
- components are meant to be composed.

It was also pointed out that components and composition are not restricted to software, but is also a term used in mathematics or hardware.

The group agreed that it does not make sense to study components in isolation, but their true value lies in their ability to be *composable.* But what kind of compositions should we consider? Putting a pair of glasses onto a table is composition, but unless we are interested in physics of molecules, no real interaction happens between the table and the glasses. This composition does not

really add any new *value* to the environment. Therefore, it was suggested that a composition of two or more components must lead to *interactions* between these components and interaction must trigger some form of *behavioural change* in at least one of the involved components. Or in other words: *interaction defines the semantics of composition.*[1]

Another approach to define composition of components and the resulting interactions is to use mathematical formalisms such as the λ-calculus and view composition as *functional composition.* Such an approach has the advantage that we can reason about composition in a well-understood framework, but the notion of a component is rather narrow and interaction can only be defined at an abstract level.

Hence, the group concluded the discussion of general terms by agreeing that *a component is an abstract view of an asset* and *assets must interact in order to be considered as components.* Although the group came up with some interesting views on components and interaction in general, it was decided to restrict our view to software assets for the remainder of the discussion.

After this initial discussion, we shifted our attention towards verification and correctness of compositions. The group agreed that in order to check the correctness of a composition, a notion of *validity* is required in the corresponding composition context. Unfortunately, most component models have no notion of validity. Hence the controversial question was raised whether we need to verify the correctness of software system at all. After a short, but intense discussion it was agreed that most software systems are verified in one way or the other and that verification is necessary.

But what does *correctness* mean in the context of composition? The correctness of a composition can be expressed both at a *syntactical* and at a *semantical* level: the former is generally associated with a type system whereas the latter is associated with some form behavioural specification. As an example, we briefly talked about the correctness of JavaBeans compositions. During this discussion, it was noted that existing languages and environments offer support for type specification and checking, but rarely support for the specification of behavioural properties of components, an area where further research efforts are needed.

Is it possible to verify the correctness of a system without considering its environment? To answer this question, it is helpful to make the distinction between a composition of components and the deployment environment of such a composed system. Explicitly considering required services of components (and the resulting composite) will probably allow us to verify the correctness of a system (at least up to a certain degree) requiring only a limited view of the environment. Therefore, it should be possible to use formal models supporting modular verification techniques and avoid problems with "full-blown" environments. This part of the discussion is probably best summarized by the following

[1] In post-workshop discussions it was suggested that this might not necessarily be true and that the semantics of composition might cover aspects that cannot be recovered from behaviour alone.

quote: "environments are important, but this does not mean that we cannot do some form of system verification without taking them into consideration."

Next, the group agreed that correctness at a syntactical level (i.e., at the level of type systems) is not enough and that behavioural specifications have to be considered. But what are suitable models to specify behaviour? It was mentioned that formal specification techniques are not yet well accepted in practice and may lead to "over-specifications." Furthermore, any kind of behavioural specification that cannot be directly expressed in source code may lead to double maintenance problems (i.e., updating both source code and the corresponding behavioural specification). Hence, there is a definite need for tool support in this area so that specification and verification can be seamlessly integrated into the development process. As the state-of-the-art is not very advanced, yet, this seems to be a promising area for further exploration.

The last topic we were able to address were suitable "tools" to specify the semantics of languages for composition. In order to properly define the semantics of such a language, it was suggested that a bottom-up approach would be the most appropriate way. As an example, the concurrent programming language Pict was briefly discussed where the semantics of higher-level language constructs is expressed in terms of a small set of primitives; these primitives can be directly mapped to constructs found in an asynchronous variant of the π-calculus.

It was also suggested that there exists more than one approach to do define the semantics of a language. Depending on the problem domain, different approaches have to be used. Furthermore, there was some consensus that defining the semantics of a language based on a small set of basic operators generally is more the exception than the rule.

One of the group members made the hypothesis that it should be possible to define a small set of basic operators suitable for defining the semantics of any programming language, but in particular any composition language. Discussing this issue, the group came to the conclusion that any approach based on a first-order formalism would be too cumbersome to use, higher-order approaches (such as higher-order logics) seem to be much more appropriate. It was also stated that higher-order approaches do not add any expressive power, but make such an approach much easier to understand. The same applies to type systems where the focus should shift towards higher-order variants.

The discussion of this breakout group concluded by stating the following question: how can we express *partial* validity of a composed system? Unfortunately, we did not have time to further discuss this issue in any detail.

Summing up the afternoon sessions it can be noted that very lively discussions were going on in both breakout groups and that various (sometimes contradictory) points of view were brought forward. Unfortunately, we were not able to discuss all topics in enough detail, but interesting results were achieved, nevertheless. Once again it has become apparent that people hold different views about the concepts related to component-based software technology in general and composition languages in particular. All participants agreed that further discussion will be needed to address these issues.

4 Conclusions

Concluding, we are able to state that the third Workshop on Composition Languages (WCL 2003) was again quite a successful event that was appreciated by all participants. As this report reveals, not all of the initial goals were met, but most importantly we were able to continue the fruitful discussions of the first two workshops on composition languages (WCL 2001 and WCL 2002) and come to consensus on some important issues.

To address the issues we have not been able to deal with during the workshop, it is necessary to further enhance and encourage research collaborations between various disciplines (such as component models and frameworks, software architectures, concepts of programming languages, model-driven development etc.) and bridge the gap between these disciplines. Some participants suggested that to define a small list of non-trivial problems suitable for exploration in the near future. Therefore, it is very likely that further events especially dedicated to the topic of composition languages will be organized at some stage.

Acknowledgments. We would like to thank all workshop attendees for their insights and useful comments as well as the local ECOOP organizers for their support.

Organizing Committee

- Markus Lumpe
  Department of Computer Science, Iowa State University
  113 Atanasoff Hall, Ames, IA 50011-1041, USA
  Tel: +1 515 294 2410, Fax: +1 515 294 0256
  Email: `lumpe@cs.iastate.edu`
- Jean-Guy Schneider
  School of Information Technology, Swinburne University of Technology
  P.O. Box 218, Hawthorn, VIC 3122, Australia
  Tel: +61 3 9214 8189, Fax: +61 3 9819 0823
  Email: `jschneider@swin.edu.au`
- Bastiaan Schönhage
  Compuware Europe B.V., Hoogoorddreef 5,
  P.O. Box 12933, 1100 AX Amsterdam, The Netherlands
  Tel: +31 20 311 6158, Fax: +31 20 311 6200
  Email: `Bastiaan.Schonhage@nl.compuware.com`
- Markus Bauer
  Forschungszentrum Informatik, University Karlsruhe
  Haid-und-Neu-Strasse 10-14, D-76131 Karlsruhe, Germany
  Tel: +49 721 965 4630, Fax: +49 721 965 4621
  Email: `Bauer@fzi.de`
- Thomas Genssler
  Forschungszentrum Informatik, University Karlsruhe

Haid-und-Neu-Strasse 10-14, D-76131 Karlsruhe, Germany
Tel: +49 721 965 4620, Fax: +49 721 965 4621
Email: Genssler@fzi.de

Accepted Papers

- Joseph R. Kiniry (University of Nijmegen, The Netherlands): *Semantic Component Composition.*
- Jørgen Steensgaard-Madsen (Technical University of Denmark, Lyngby): *Composition by continuations.*
- Nigel Jefferson and Steve Riddle (University of Newcastle upon Tyne, UK): *Towards a Formal Semantics of a Composition Language.*
- I.S.W.B. Prasetya, T.E.J. Vos (Universidad Politécnica de Valencia), S.D. Swierstra (Utrecht University, The Netherlands), and B. Widjaja (University of Indonesia): *A Theory for Composing Distributed Components, Based on Temporary Interference.*
- Naiyana Tansalarak and Kajal T. Claypool (University of Massachusetts, Lowel, USA): *XCompose: An XML-Based Component Composition Framework.*
- Arne Berre (SINTEF, Oslo, Norway): *Support for Model-centric and Architecture-centric Development with workflow based composition.*
- Andreas Heberle (Entory AG, Ettlingen, Germany) and Rainer Neumann (PTV AG, Karlsruhe, Germany): *Model Driven Development of Component Centric Applications.*
- D. Janaki Ram and Chitra Babu (Indian Institute of Technology Madras, Chennai, India): *ChefMaster: an Augmented Glue Framework for Dynamic Customization of Interacting Components.*

Related Links

- The workshop web-site:
  http://www.cs.iastate.edu/ lumpe/WCL2003/
- WCL 2002 workshop web-site:
  http://www.cs.iastate.edu/ lumpe/WCL2002/
- WCL 2001 workshop web-site:
  http://www.cs.iastate.edu/ lumpe/WCL2001/
- The Piccola language page:
  http://www.iam.unibe.ch/ scg/Research/Piccola/
- The Pict programming language:
  http://www.cis.upenn.edu/ bcpierce/papers/pict/Html/Pict.html
- Unified Modeling Language (UML):
  http://www.uml.org
- Object Management Group (OMG):
  http://www.omg.org
- OMG Model-Driven Architecture (MDA):
  http://www.omg.org/mda/

List of Participants

Markus Bauer	Forschungszentrum Informatik Karlsruhe, Germany Bauer@fzi.de
Steffen Becker	University Oldenbourg, Germany becker@informatik.uni-oldenburg.de
Arne-Jorgen Berre	Sintef/University of Oslo, Norway Arne.J.Berre@sintef.no
Robert Bialek	University of Copenhagen, Denmark bialek@diku.dk
Thomas Genssler	Forschungszentrum Informatik Karlsruhe, Germany Genssler@fzi.de
Jon-Yun Jung	Ajou University, South Korea jongyun@ajou.ac.kr
Hak-Min Kim	Ajou University, South Korea hakmin@ajou.ac.kr
Kee-Hyun Kim	Ajou University, South Korea misozigi@ajou.ac.kr
Joseph Kiniry	University of Nijmegen, The Netherlands kiniry@acm.org
Markus Lumpe	Iowa State University, USA lumpe@cs.iastate.edu
Fernando Lyardet	Technical University Darmstadt, Germany fernando@tk.informatik.tu-darmstadt.de
Rainer Neumann	PTV AG, Germany Rainer.Neumann@ptv.de
Jacques Noyé	Ecole des Mines de Nantes, France Jacques.Noye@emn.fr
Wishnu Prasetya	Utrecht University, The Netherlands wishnu@cs.uu.nl
Alexander Romanovsky	University of Newcastle upon Tyne, UK alexander.romanovsky@ncl.ac.uk
Jean-Guy Schneider	Swinburne University of Technology, Australia jschneider@swin.edu.au
Bastiaan Schönhage	Compuware Europe B.V., Amsterdam, The Netherlands Bastiaan_Schonhage@nl.compuware.com
Jørgen Steensgaard-Madsen	Technical University of Denmark, Denmark jsm@imm.dtu.dk
Andrew Wils	K.U. Leuven, Belgium andrew.wils@cs.kuleuven.ac.be
Naiyana Tansalarak	University of Massachusetts, USA ntansala@cs.uml.edu

Tools and Environments for Learning Object-Oriented Concepts

Isabel Michiels[1], Jürgen Börstler[2], Kim B. Bruce[3], and Alejandro Fernández[4]

[1] PROG, Vrije Universiteit Brussel, Belgium
[2] Umeå University, Sweden
[3] Williams College, Massachusetts, USA
[4] Fraunhofer IPSI, Darmstadt, Germany

Abstract. This report summarizes the results of the seventh workshop in a series of workshops on learning object-oriented concepts. The focus of this workshop was on (computer-aided) support for the teaching and learning of basic object-oriented concepts.

1 Introduction

Teaching the object-oriented paradigm in the same way as "traditional" introductory programming courses does not appear to work very well. In the pre-OO world, concepts could be introduced step by step and the grouping of program elements could be handled as an afterthought. There was no need to introduce high-level and abstract structures, like modules or abstract data types, early on. This is very different in the object-oriented paradigm, where the basic concepts are tightly interrelated and seem not to be easily explained in isolation. Instead they need to be handled in groups (like, for example, variable, value, type, object, and class) making teaching and learning more challenging. Grasping the "big picture" may furthermore be hindered by focusing on notational details of specific object-oriented programming languages. Most students therefore have difficulties taking advantage of object-oriented concepts.

This was the seventh in a series of workshops on issues in object-oriented teaching and learning. Reports from all previous workshops in the series are available [1,2,3,4,5,9]. Further information can be found at the workshop series home page [17].

The objective of the workshop series was to share experiences and discuss ideas, approaches and hypotheses on how to improve the teaching and learning of object-oriented concepts.

Each workshop in the series focussed on a specific theme. This workshop's theme was (computer-aided) support for the teaching and learning of basic object-oriented concepts. The organisers particularly invited submissions on the following topics:

- intelligent learning environments (for teaching object technology)
- frameworks/toolkits/libraries for learning support
- approaches and tools for teaching design early

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 119–129, 2004.

- different pedagogies
- design early vs. design late
- frameworks/toolkits for the development of teaching/learning applications
- experiences with innovative CS1 curricula
- usage of metaphors, analogies, and illustrative examples
- distance education

2 Workshop Organization

Participation at the workshop was by invitation only. The number of participants was limited to encourage the building of few small interest groups working on specific topics. Potential attendees were required to submit position papers.

Out of the 17 position papers that were submitted, six papers were selected for presentation at the workshop. The authors of nine other papers were invited for participation at the workshop. All contributions were made available on the workshop's website a few weeks before the workshop, in order to give attendees the possibility to prepare the discussions. Presentations were scheduled for the morning sessions. The afternoon sessions were dedicated to discussions in small working groups.

All attendees were given the opportunity to present their positions briefly (one overhead, at most two minutes time). After that three working groups were formed to discuss in more detail the topics they found most relevant.

The workshop gathered 28 participants from 11 different countries, all of them from academia. A complete list of participants together with their affiliations and e-mail addresses can be found in table 1.

3 Summary of Presentations

This section summarizes the main points of the presented papers and the most important positions of the other participants. More information on the presented papers can be obtained from the workshop's home page [10].

Uwe Thaden (Learning Lab, Hannover, Germany) discussed the usage of 3D animations of UML diagrams to visualize the execution of object-oriented code.

A basic premise for good software development is the ability of programmers to think in the concepts of the programming language used. Thus, today object-oriented programming needs to be taught in an adequate way. The procedural or imperative style of programming corresponds very well with the verbal descriptions of algorithms, concentrating on control flow. Important and typical for the object-oriented paradigm are the dynamic creation and destruction of objects, the links between them, and their interaction through exchanging messages. Program execution is distributed over several objects.

Object-oriented programming was conceived as a way of modeling reality. It seems to be worth examining whether this naturalness can be exploited to explain program execution other than by mapping it to a sequence of instructions to a register machine. The Unified Modeling Language (UML) offers several dynamic

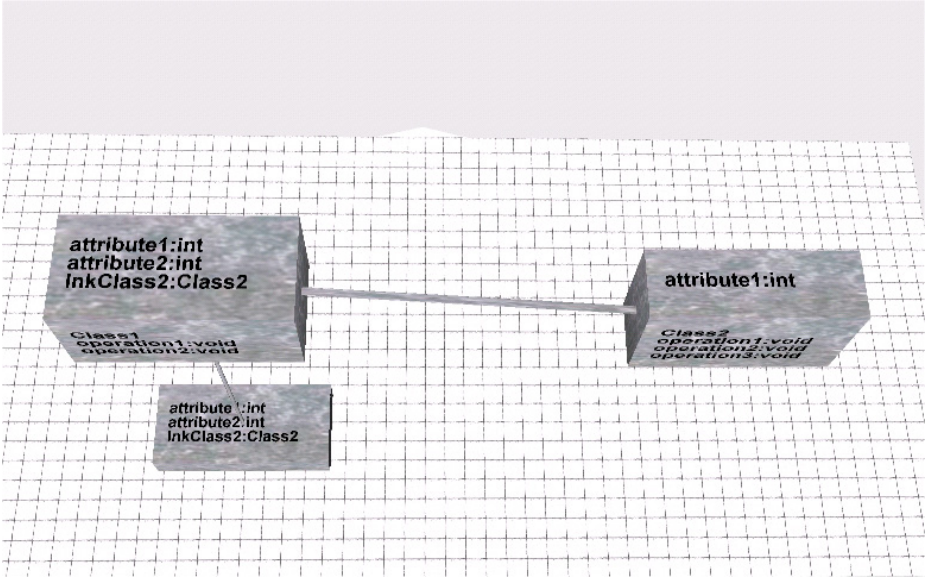

Fig. 1. Example view of an instantiation

diagram types (e.g. sequence- and collaboration diagram) of particular interest for such an approach.

However, paper-bound visualizations of program executions are inevitably static projections that cannot transport program dynamics. The goal of the presented approach is not only to visualize algorithm execution, but to develop a tool to aid the intuitive understanding of an arbitrary Java program execution. The presented (Java) prototype uses an XML structure representing UML diagrams. This structure can be obtained from existing Java programs by means of UML modeling tools ("Together" in this case). The XML structure is then transformed into VRML that can be displayed in web browsers.

Figure 1 shows a screenshot taken from an instantiation animation. The classes in this example are shown at a higher level while the instances are placed on the "ground" to visualize the conceptual difference between class and object in an intuitive way.

Jörg Pleumann (University of Dortmund, Germany) proposed a framework for lightweight object-oriented design tools. Modeling on the basis of formal visual languages like the UML has become a central activity in today's software development processes. Universities need to teach new software developers how to express their ideas in the form of models and how to understand the models created by others. When the size of models approaches that of non-trivial problems, tool support becomes crucial – even during education. Unfortunately, the typically-used industrial CASE tools have major drawbacks when applied in an educational setting. Aimed at professional work, they are too complex for

classroom-use and lack features that support their users in learning a modeling language.

The Java-based framework for so-called "lightweight" modeling tools developed at the University of Dortmund is meant to be a first step towards didactical modeling tools. All tools derived from the framework share the same simple and intuitive user interface and have only small resource demands. The two main extension points of the framework are a generic metamodel and generic graphical figures. New tools are easily developed by "plugging in" the concrete metamodel and graphical elements of a specific modeling language. Three modeling tools have already been built using the framework; a statechart editor and simulator, a software architecture editor and a toolset for the Unified Process. The statechart editor features a multimedia simulation engine interfacing with the user's model. This interface can be used to control a washing machines or a coffee machine. The tool has already been used successfully in a software engineering class at Dortmund.

Jürgen Wolff von Gudenberg (University of Würzburg, Germany) claimed that programming can only be learned by reading and, first of all, by writing programs. Hence, an online tutorial should provide a high level of interaction. The evaluation of and immediate response to performed tests or submitted programs enhances the value considerably.

The presented JOP (Java Online Praktical) was developed by the universities of Passau and Würzburg (both Germany). It comprises two parts: a Java tutorial and evaluator to check exercises and give feedback to students.

The tutorial is organized as a collection of learning units together with immediately executable and modifiable examples and exercises. It can be noted that interfaces are discussed long before inheritance and polymorphism. I/O and GUIs are the last topic. Examples and exercises are evaluated by electronic tutors. These tutors perform functional as well as structural tests and provide immediate feedback to the users. Examples of tests performed are conformance to coding conventions, proper documentation, usage of required/forbidden classes and the correct execution of supplied functional tests.

The system has been used in different lectures. In those lectures about 2500 programs for 16 different problems have been evaluated. The average size of a program was about 1000 lines of Java code. The work load for examining and marking the programs could be decreased by a factor of four, while the quality and readability of the programs increased significantly. The tests are furthermore quite reliable. Only about two percent of the programs that passed all automatic tests were not accepted by a human inspector.

Ola Berge (Intermedia, University of Oslo, Norway) talked about socio-cultural perspectives on object-oriented learning and how these aspects relate to the COOL (Comprehensive Object-Oriented Learning) project [13]. One of the central objectives of this project is to explore critical aspects associated with learning (and teaching) object-oriented concepts. This objective will be pursued by bringing forth the heritage of Kristen Nygaard and the Norwegian approach to object-orientation. The project aims at developing learning materials based on

this work and intends to make those materials available to the object-oriented community in the form of reusable learning objects.

Berge and colleagues apply a socio-cultural perspective on the area of learning object-oriented concepts. It raises a number of issues concerning design of learning objects and learning environments for knowledge construction in this field. The COOL project explores these issues by studying current practices as well as by experimenting with new constellations of artifacts. Early activities include the development of courses that introduces learners and practitioners to fundamental object-oriented concepts through approaches such as "models first" and "objects first." The courses will be developed evolutionary, where experience from the first courses, planned to be held in Oslo in the summer of 2003, will provide insights for further improvement of the subsequent activities.

The socio-cultural perspectives give directions of how tools (e.g. KarelJ [15] or BlueJ [12]) and other ICTs (Information and Communication Technologies) should be incorporated in the learning activity. Therefore, questions arise on how existing and new learning material should be implemented as learning objects in ICT-based learning environments. Last, but not least, these perspectives provide also guidelines on how to understand the meaning of social interaction with respect to learning.

Virginia Niculescu (Babes-Bolyai University, Cluj-Napoca, Romania) presented an approach to teach creational design patterns based on simple algebraic structures.

Teaching design patterns to students with little experience in OOP is difficult. It is therefore crucial to use examples from well-known domains. Since most CS programs start with (among others) Algebra, this seems a suitable area.

To implement a general algebraic structure over a field of special values, such as null and unity elements, are needed. To build a structure that is independent from the specific field chosen, these special values must be created by specific methods. The goal is to develop a general polynomial class that is independent from the type of its coefficients, i.e. uses an abstract coefficient type (`FieldElement`). When working with polynomials over specific coefficient types it is necessary to handle special values independent of their exact type. For instance, when one constructor of the class polynomial creates a null polynomial, one has to create a null coefficient, even if its exact type is not known.

Niculescu presented three approaches to solve this problem, based on three classic creational design patterns [6]: Factory Method, Abstract Factory, and Prototype. Students can then compare solutions to specific problems using all three approaches. This enables students to understand these design patterns, and the differences between them. Experience confirms that applying patterns in a known domain helps students to better understand the concepts. In the future this work will be extended to cover further algebraic structures, such as matrices, and to build a general algebraic library. This might even help to reinforce some of the mathematical concepts.

Arne-Kristian Groven (Norwegian Computing Centre, Oslo, Norway) argued that OO programming should be regarded as modeling, establishing the ba-

sic concepts of the OO perspective early in the study. The dominating current pedagogical approach is often justified by the statement that "teaching must start with sufficiently simple examples." This statement is often (mis-) understood as to use traditional examples developed with the imperative programming paradigm in mind.

The Scandinavian tradition builds upon "sufficiently complex examples" as propagated by Kristen Nygaard. Modeling is about creating a description of phenomena and concepts from a given application domain. In OO modeling these phenomena and concepts are described as classes and objects. A model is itself an abstraction of something for the purpose of understanding it. The expressiveness of programming languages is limited. They are therefore not suitable to describe all aspects of an application domain. It is therefore imperative to expose students to different programming languages to enrich their vocabulary for modeling different aspects of the application domain.

Groven and colleagues have started to monitor and evaluate OO courses to identify recurring problems and approaches that work or do not work respectively. As a next step they have planned to carefully design interventions in existing courses to study the effects of modeling based approaches. The first results are expected for the beginning of 2004.

4 Working Group Discussions

For the afternoon sessions participants formed three working groups to discuss the following topics. The following subsections summarize the results from these workings groups.

1. Tools
2. OO languages for teaching - why all this Java?
3. What are the real hard problems?

4.1 Tools

This group discussed the usage of tools in various teaching situations. Four main areas for tool support were identified; modeling and design, coding, execution, and evaluation. The strengths and weaknesses of several popular tools were shortly discussed (Alice [11], BlueJ [12], Fujaba [14], JKarel [15], and Praktomat [16]). It was noted that all of them worked excellently in at least one of the four areas above, but none of them did support all four areas.

Working group participants would require the following minimum properties/capabilities from a good OO educational tool:

- clear purpose
- simple & lightweight, but not simplistic
- impose restrictions on programming styles
- visualization of collaborations between objects
- easy transition to "real life" programming tools or environments

Finally, the group expressed also what they wish to see more of in available or future tools:

- traceability; seeing the effects of change
- better integration of tools with each other
- student progression monitoring
- (semi-) automatic evaluation of models, code, and documentation

4.2 OO Languages for Teaching – Why All This Java?

This group debated on the use of OO languages for teaching purposes. The following three questions came to the surface at the beginning of the discussion:

1. Do we need a subset of an existing language?
2. Do we need a brand new language?
3. Does the perfect language perhaps already exist?

Group members agreed on the value of the following language features to support learning of OO:

Static Types and Type Declarations. The discussion emphasized that the following properties of static type systems were often valuable in teaching novices:

- safety
- documentation
- tests (powerful documentation!)
- programming environment
- type inference
- beta patterns

Closures. There was some discussion as to whether it was useful to have a language that provided support for closures. At least one participant, Bruce, argued that they were not necessary, and that some of the same expressiveness could be provided in a language like Java with inner and anonymous classes. Even then these features were not necessary in a first semester course, though inner classes can be quite helpful in a course on data structures.

Don't fall off a cliff. The participants agreed that it could be helpful to use programming environments that provide support for focusing on only parts of the programming language (the part that students are learning at some time). Students shouldn't accidentally use part of a language or an environment which they should not know (yet), in order to avoid confusion. In DrScheme for example, teaching packs are offered depending on the level of the student.

4.3 What Are the Real Hard Problems?

The following aphorisms, ideas, and questions came up during the discussion:

- Is it hard to teach or to learn?
- We need to bridge the gap between stories and the program
- Students are not empty bowls
- Students can learn by seeing other students' errors
- As with writing poetry, students should learn to read before learning to write
- A good programmer is lazy – they copy the best practice
- Good design comes from experience – experience comes from bad design

This group started the discussion with a comment of Joe Bergin. He said that if you teach right, learning is easy. But since we still do need to figure out how to teach right, we tried to identify the current problems we have.

Abstraction was considered to be one of the hardest problems – to let students grasp the object-oriented way of thinking and to learn how to generalize a problem. Teaching more than one paradigm was found to be important for broadening a student's mind. Using functional programming was mentioned in this context, because for example Scheme, a dynamically typed, functional language, is an excellent tool for learning about abstraction. Consider, for example, defining higher-order functions in Scheme. This way, you learn to factor out the variabilities of common subproblems.

Polymorphism is a tough topic as well, although others find it a quite natural topic if you succeed in relating polymorphism with things that are already known in the real world. An example of this could be asking a question to two different people, as they might respond in completely different ways.

Another hot topic in this group was a more general remark: other students can benefit by seeing other students' errors. Student assignments based on this idea were discussed, like grading students on their ability to criticize the other's designs, and more than that, also grade them on their ability to improve their work after hearing the criticism.

Another range of hard problems are the ones related to design and modeling: how do you visualize things for students, and how do you provide them with the sort of mental model they need to have? It is clear that even for experienced practitioners, everyone uses their own way of mentally drawing their idea of the design, and even if people use the same notation, people sometimes assign different semantics to it.

5 Conclusions

The objective of this workshop was to discuss current tools and environments for learning object-oriented concepts and to share ideas and experiences about the usage of computer-based tools to teach the basic concepts of object technology.

Ongoing work was presented by a very diverse group of people with very different backgrounds, which resulted in the discussion of a broad range of topics like tool support, environments, courses, teaching approaches, languages for teaching, etc.

Table 1. Workshop participants

Name	Affiliation	E-mail Address
Isabel Michiels	*Vrije Universiteit Brussel, Belgium*	imichiel@vub.ac.be
Jürgen Börstler	*Umeå University, Sweden*	jubo@cs.umu.se
Kim Bruce	*Williams College, USA*	kim@cs.williams.edu
Alejandro Fernandez	*Fraunhofer IPSI, Darmstadt, Germany*	casco@ipsi.fhg.de
Ola Berge	*Intermedia, University of Oslo, Norway*	ola.berge@intermedia.uio.no
Joe Bergin	*Pace University, USA*	jbergin@pace.edu
Andrew P. Black	*OGI School of Science and Engineering, Oregon, USA*	black@cse.ogi.edu
Richard Edvin Borge	*University of Oslo, Norway*	richared@ifi.uio.no
Thomas Cleenewerck	*Vrije Universiteit Brussel, Belgium*	tcleenew@vub.ac.be
Pedro J. Clemente	*University of Extremadura, Cáceres, Spain*	jclemente@unex.es
Jessie Dedecker	*Vrije Universiteit Brussel, Belgium*	jededeck@vub.ac.be
Wolfgang De Meuter	*Vrije Universiteit Brussel, Belgium*	wdmeuter@vub.ac.be
Arne-Kristian Groven	*Norwegian Computing Centre, Oslo, Norway*	Arne-Kristian.Groven@nr.no
Håvard Hegna	*Norwegian Computing Centre, Oslo, Norway*	havard.hegna@nr.no
Mario Kusek	*University of Zagreb, Croatia*	mario.kusek@fer.hr
Dennis Medzihradszky	*Dennis Gabor College, Budapest, Hungary*	medzihradszky@ szamalk.hu
Birger Møller-Pedersen	*University of Oslo, Norway*	birger@ifi.uio.no
Marie-Helene Ng	*Birkbeck College*,	gngch01@dcs.bbk.
Cheong Vee	*University of London, UK*	ac.uk
Virginia Niculescu	*Babes-Bolyai University, Romania*	vniculescu@nessie.cs.-ubbcluj.ro
Jörg Pleumann	*Universität Dortmund, Germany*	pleumann@ls10.cs.uni-dortmund.de
Wilfried Rupflin	*University of Dortmund, Germany*	wilfried.rupflin@uni-dortmund.de
Jens Schröder	*Universität Dortmund, Germany*	schroeder@ls10.cs.uni-dortmund.de
Tyszberowicz Shmuel	*Tel-Aviv University, Israel*	tyshbe@post.tau.ac.il
Marianna Sipos	*Dennis Gabor College, Budapest, Hungary*	sipos@szamalk.edu
Friedrich Steimann	*Learning Lab, Hannover, Germany*	steimann@acm.org
Uwe Thaden	*Learning Lab, Hannover, Germany*	thaden@learninglab.de
Jürgen Wolff von Gudenberg	*University of Würzburg, Germany*	wolff@informatik.uni-wuerzburg.de
Amiram Yehudai	*Tel-Aviv University, Israel*	amiramy@tau.ac.il

Summarizing what was said in the debate groups, we can conclude that:

- Teaching/learning tools should be lightweight and reflect a clear purpose. Support should be provided for visualizing the different aspects of the OO paradigm like object interactions. Integration of different educational tools should be increased, and the transition to *real-life* programming should be easily possible.
- There are many advantages to using an OO language for teaching that supports static types and type declarations. It is also very helpful if support is provided for focussing only on some aspects of the language (concepts that don't have to be known yet shouldn't be made available) - like teaching packs in DrScheme.
- Having talked about what the hard problems in teaching are, we concluded that abstraction(learning to generalize), polymorphism, teaching design, and providing mental models provide the biggest problems.
- Getting to know different programming paradigms is important; other paradigms, like functional programming, can aid significantlyin clarifying different aspects of programming.

References

1. Börstler, J. (ed.): OOPSLA'97 Workshop Report: Doing Your First OO Project. Technical Report UMINF-97.26, Department of Computing Science, Umeå University, Sweden (1997).
2. Börstler, J. (chpt. ed.): Learning and Teaching Objects Successfully. In: Demeyer, S., Bosch, J. (eds.): Object-Oriented Technology, ECOOP'98 Workshop Reader. Lecture Notes in Computer Science, Vol. 1543. Springer-Verlag, Berlin Heidelberg New York (1998) 333-362.
3. Börstler, J., Fernández, A. (eds.): OOPSLA'99 Workshop Report: Quest for Effective Classroom Examples. Technical Report UMINF-00.03, Department of Computing Science, Umeå University, Sweden (2000).
4. I. Michiels, J. Börstler: Tools and Environments for Understanding Object-Oriented Concepts, ECOOP 2000 Workshop Reader, Lecture Notes in Computer Science, LNCS 1964, Springer (2000), pages 65-77.
5. I. Michiels, J. Börstler and K. Bruce: Sixth Workshop on Pedagogies and Tools for Learning Object-Oriented Concepts, in J. Hernández and A. Moreira, editors, Object Oriented Technology - ECOOP 2002 Workshop Reader, Volume 2548 of Lecture Notes in Computer Science, LNCS 2548, Springer (2002), pages 30-43.
6. E. Gamma , R. Helm , R. Johnson , and J. Vlissides . Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
7. Pedagogical Patterns pages. `http://www-lifia.info.unlp.edu.ar/ppp/` `http://csis.pace.edu/~bergin/PedPat1.3.html`
8. European Master in Object-Oriented Software Engineering. `http://www.emn.fr/MSc/`
9. OOPSLA01 workshop. `http://www.cs.umu.se/%7Ejubo/Meetings/OOPSLA01/`
10. ECOOP 2003 Workshop homepage. `http://prog.vub.ac.be/~imichiel/ecoop2003/workshop/`
11. Alice homepage. `http://www.alice.org`

12. BlueJ - the interactive Java environment. http://www.bluej.org
13. COOL project Workshop homepage. http://www.intermedia.uio.no/cool/
14. Fujaba (From UML to Java And Back Again) homepage. http://www.uni-paderborn.de/cs/fujaba/
15. JKarel Robot homepages. http://csis.pace.edu/~bergin/#kjr and http://math.otterbein.edu/JKarelRobot/
16. Praktomat. http://sourceforge.net/projects/praktomat/
17. Workshops on OO Education. http://www.cs.umu.se/research/education/ooEduWS.html

Patterns in Teaching Software Development

Erzsébet Angster[1], Joseph Bergin[2], and Marianna Sipos[1]

[1] Dennis Gabor College, Programming Department, Etele ut 68.
1115 Budapest, Hungary
{angster, sipos}@szamalk.hu
http://www.gdf.hu/progtanszek/angster, sipos
[2]Pace University, Computer Science, One Pace Plaza,
New York, NY 10038, USA
berginf@pace.edu
http://csis.pace.edu/~bergin

Abstract. This paper summarizes the results of Workshop #14 held at ECOOP 2003. The goal of this one-day workshop was to discuss ideas on what and how to teach Software Development. The workshop focused on using patterns and concrete examples. Participants were encouraged to submit a small but concrete project (documentation and the running application). Based on the project, workshop presenters were encouraged to submit a paper including fields like: Teaching environment, Important teaching areas, Pedagogy, Feedback from industry etc. As a main topic, participants discussed, that there are not enough concrete examples made public, which could help in software development.

1 Introduction (Aim, Organization)

The goal of this one-day workshop was to discuss ideas on what and how to teach Software Development. **Participants were encouraged to submit a small but concrete project** (documentation and the running application) developed by the students. Based on the project, workshop presenters were encouraged to submit a paper including the following:

- Project's teaching environment, such as type of exercise, required knowledge (preconditions), time to solve, teamwork etc.
- Main points you teach and expect your students to do when developing a system?
- Important teaching areas, such as: Design; Languages; Environments; Frameworks; Patterns; WEB-based applications; Large and distributed systems; Methodologies, including agile methodologies ...
- Pedagogy: How to teach; When and how to introduce topics; Teamwork ...
- Feedback from industry ...

The following questions were in focus, relating to software development teaching: What is more important? Teaching the most accurate technology or giving a sound

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 130–142, 2004.

basis? What concepts and methodologies should we teach? What pedagogy works for which topics? What are the best environments/languages/frameworks to use?

On the workshop however there was an extra question, on which the debate mostly concentrated: „Can we find suitable teaching materials, software pattern collections, and concrete examples for teaching Software Development?"

During the workshop participants filled out a survey about Software Development Learning Support. The survey wanted to measure, whether teachers are satisfied with Software Development materials, the public pattern collections and concrete examples.

In the following, first we give brief summaries of workshop presentations, then the summary of the workshop discussion.

2 Position Paper Summaries

On the workshop's website [WS14] you can find the position papers and additional information.

2.1 Besides SD Patterns, Concrete Examples Are Needed! (Erzsébet Angster, Dennis Gabor College, Budapest)

Software patterns are very popular. Many books, conferences, web pages and forums are about patterns to help software development (SD). The pattern community invests a considerable amount of energy to develop software patterns. However developers, especially teachers and students still find the development and documentation process very difficult. I suggest several reasons for the problem:

1. The best patterns are confidental, they are not public.
2. The catalogues are not properly arranged;
3. There are no concrete and complete SD examples (documentatation and running application) on the market. People do need the concrete examples.

Buschmann at al. say in [POSA], p422: „... most domain-specific patterns are confidential – they represent a company's knowledge and expertise about how to build particular kinds of applications, so references to them are not available. We believe however, that more and more of this knowledge will become public over time. **In the long term, sharing experience is usually more effective for everyone than trying to hold onto secrets**." My opinion is the same, with the further addition that not only the patterns must be made public but the concrete examples as well. And all of them must be well-organized.

I think, that within the SD and pattern community there are two sub-communities, who use patterns in different ways: **Developers** develop software. After years they gain expertise, so new applications are getting better and better. Inside the same workplace developers share their experience, and gather their solutions to the recurring problems. They make an abstraction of the problem in question, so that next time

they can use it in similar situations. In this way, developers produce patterns (at least they have the chance to do so), but they do not have the urge to make it public**.** They also have their concrete examples behind these patterns, but these are company secrets. In contrast to developers, **teachers and students** get information from published materials. Since they will never have a real developers' expertise, they especially need the patterns and the concrete examples with them.

Concrete and complete software development examples are needed! Claims for concrete examples are not new. There are several available exercise books, program collections on the market. The problem is, that they are not practical enough and you cannot find a complete, marketable software with documentation, even a small one.

A possible solution – a website with Software Development Packs (SDP): As there is „Pattern home page", there could be „Complete software examples home page" as well. Since I am convinced that many teachers, students and developers would be happy to use it, I intend to create one. The page would contain shepherded SDP-s. An SDP (Software Development Pack) would contain the following things: problem definition, technical documentation, user's guide, source code and the running software. Optionally, it can contain the model file, development history, used patterns and teaching material. On the website there could be an advanced search on different package features: development process, notation, programming language, problem category, pattern, size etc.

Paper: [WS14], Angster_Concrete.pdf

2.2 Lost in Object-Space (Thomas Cleenewerck, Vrije Universiteit, Brussel)

Modeling an application using small units of representation and behavior increased the reusability of various program parts and evolvability of the whole program. Moreover it has been the focus of much future research. However this 'progress' led to more and more complex solutions and designs. Various models were introduced to keep an overview of the whole system. The most important ones are the class models and the collaboration models. The class model is a static model representing static information. The collaboration model however, is a static model of dynamic behavior. When teaching object orientation to students, we are faced with the *fragmentation* and the *complexity* of the programs and the lack of an adequate model for explaining or capturing the *dynamic* behavior of objects in a program. We refer to this state of mind with the verb phrase "*lost in object-space*". To tackle these problems an environment is built that visualizes the execution of a program. Its main goals are to illustrate the interaction of objects and the role the objects play within a program. The difference with most environments is that the visualization not merely intended for illustrating object oriented programming concepts and that the visualization techniques are based on a lot of human computer interaction research performed over the last decade. The latter is achieved by drawing a parallel between object oriented programs and hypermedia systems. We observed that the lost in object-space problem is quite similar to the *lost in hyperspace* problem in which the user is lost in the structure of the web-application. By using the solutions for the lost in hyper-space prob-

lem, we were able to build a visualization tool which gave the students enough context and clues so that they are able to understand in which point in the execution the program is currently and thus to really tackle the lost object-space problem.

Paper: [WS14], Cleenewerck_Lost.pdf

2.3 Projectory (Jens Gößner & Friedrich Steimann & Thomas Mück, University of Hannover)

Java courses are part of most contemporary computer science curricula. However, no matter how good the theoretical training is: programming is best learnt by doing, i.e., in software practicals. Unfortunately, organizing and carrying out these practicals is highly labour intensive, and the outcome depends to a large extent on the attention paid to the individual student. Making this process more efficient, an IDE-integrated teaching system, called Projectory, was developed at the University of Hannover to be used in the software practical regularly conducted in our applied informatics curriculum.

Our intention was to create a programming environment enabling a constructivist approach for distance education. Such a framework should relieve the teacher from approving the correctness of solutions (marking), giving him more time for helping the students with their efforts to find a solution. This can be accomplished by the creation of a constructivist learning environment, which allows the students to experiment with possible solutions, find out how they failed, and try alternatives.

Projectory is an environment which tries to unite these demanded qualities. The intelligent testability of the student's solutions is a prerequisite for a constructivist approach. With every block of exercises a suite of test cases will be delivered to the student. A solution will be tested automatically with these test cases. In the case of an error the test suite supplies the location of the error, a message what is wrong in the solution and a hint towards a better solution.

With the Projectory-Framework we are able to perform programming courses with different content. An author must deliver an initial project skeleton, the exercises divided in blocks and formulated in HTML, a sample solution for each block and the test suites for each block.

Projectory supports distance working distributed collaboration. The exercises are provided at a remote server. Every student downloads the current block of exercises and test cases from this server and submits his solution to this server, all from within the integrated development environment supplied with Projectory.

The automatic verification of the submitted solution is performed both locally and remotely. Local verification gives the students the facility to get immediate feedback on how they are doing. Upon completion of a block of exercises, the student uploads its work to the server. All tests must pass to confirm the correctness of the solution. If tests fail the server generates a report for the user.

The use of Projectory results in a smooth and closed process from the downloading of exercises until the submission of the solutions. Projectory increases the efficiency of the students and reduces the teacher's role to a tutor for the students. However, the

use of Projectory leads also to relatively great effort as regards the preparation of exercises. This concerns primarily a more precise formulation of exercises, the constructions of test suites and the accompanying example solutions for each single block.

Paper: [WS14], Goesner_Projectory.pdf

2.4 Teaching Reliability and Schedule Control (Krzysztof Kaczmarski, Warsaw University of Technology)

Problems in Teaching Software Development. In most of the projects developed by students during studies there is no cost/time control. Participants often just choose a project, and then must develop it. They usually write applications at home or in university labs but they do not control how much time they spend working on it. They independently organize their time, so we actually do not know if they make good time/organization decisions. After some time students simply bring finished (working or not) applications and present documentation.

Our investigations (and self experiences) proved that students often spend a lot of time on not important parts of the project, noticing real difficulties in the most unwanted moment, which led to unexpected problems and time delays. They also usually made some accidental movements which often did not help with certain problem.

One of the most important goals of our teaching pattern is to force students to control their time – use systematic time schedule forecasting, and methodology – choose and follow one. A good students' system design becomes a basis for the whole time schedule.

Development Control Teaching Pattern. The pattern is intended to break the impossibility in schedule and development control. The goal is to teach how the development process should be organized. It is quite obvious that we can not just assign the task and then wait for results. This is not going to work unless someone will show students' where there made mistakes and how they should perform certain tasks.

Our pattern is based on constant training which is done during each meeting. Thus the pattern could be called: "Every Day Students Progress and Development Control". The educator instructs students on

- development process: how to organize the project, how to control the progress;
- efficient team work: how to organize and share the code repository, how to organize document flow, how not to break working system.
- assuring reliability: how to organize tests, how to maintain the development environment, how to refactor

He should also perform several checks to assure that students are following the right path. It should at least consist of timetable, development process control. There should also be an interview performed with a leader of each team.

We believe that in the most cases the only way to help students in achieving these goals are practical laboratories joined with individual consultation repeated periodically. An educator analyzes the situation and helps students if such help is needed. These process must be performed all the time when students work. Final remarks on

the end of the development process, which are usually given at standard labs are much too late. We believe that guided students will learn much more if they try to follow one required path from the beginning.

Paper: [WS14], Kacz_Reliability.pdf

2.5 Students' Contributions to Software Development Teaching (Dénes Medzihradszky, SZÁMALK, Budapest)

Teaching object-oriented software development (SD) calls for special thinking. Objects alone would not fulfill the special needs a software developer has to face thus patterns were brought into consideration in the mid-nineties. Patterns are defined in one aspect as a relation between a certain context, a problem and a solution. General applications can be created by using well established patterns which would serve as examples for the developmental work. In contrast to the available abundant literature about patterns in general, little has been published about concrete examples. Using a software development competition organized annually we have tried to establish certain patterns for general use based on the more usual applications contributed. One type of application was selected an analyzed to find the common elements and deduce the pattern behind that typical application.

If we look at the applications we use in general, there are high degree similarities between applications developed for a given purpose. Thus patterns may be looked upon as a higher organization level, an example, how to solve a typical programming task to fulfill the general requirements of that task. Although much was written about patterns, practically no examples can be found in the literature or over the Internet to help in teaching. To establish suitable examples, we have two different approaches, either the synthetic method, to build patterns as required upon theoretical considerations or the analytical method, when we might study already written applications to generalize the features which would form a pattern. The latter might be looked upon as a **data mining approach**, where we made use of a competition organized annually by Erzsebet Angster and Istvan Selmeci of the Dennis Gabor College to create software development examples useful for teaching. We have selected the Address Book problem (the users can store and search for names, addresses, phone numbers, and pictures as well) and from the more than twenty submitted solutions using data mining we were able to identify pattern examples for Personal Data Entries, for Group Registration and Assignment but the database structure and the application itself as well can be looked upon as pattern examples themselves. We came to the conclusion, that by selecting the best solutions for a given problem plus generalizing these solutions and highlighting the common features might serve as excellent sources for badly needed pattern example teaching materials.

Paper: [WS14], Medzi_Students.pdf

2.6 Teaching Object-Oriented Development – 4GL Tools – Programming Patterns (István Selmeci, SZÁMALK, Budapest)

As a teacher of SZAMALK Education Ltd Hungary, I lead regular and evening / distance programming trainings. My experiment is that students acquire curriculum only with difficulty.

Functionality of the computers is based upon formal logic and actions must be fully defined before their execution. Programs have to contain every details that are needed in the solution of the given task, including data definitions and transaction specifications. A good programmer has to "think and work" like the computer itself.

People do not act as logical creations. Originally the human ways of thinking is not logical, it is rather intuitive, "jumping", convergent and divergent at the same time. When thinking, we rarely take care of details – there are many things, that we think of "given, naturally existing". When I teach beginners, I see problems that I think are derived from this inconsistency. How could we dissolve this inconsistency between the computers' work and human thinking?

Traditional training begins with the details: elements of programming languages, "Hello world" style tasks, meaningless simple algorithms – and additionally, computer-like thinking is expected at once. In case of heterogeneous audience, the result is also heterogeneous.

Nowadays, we are using modern 4GL tools, and the situation is not better. The graphical IDE makes the mechanism mysterious and the RAD philosophy enforces its will onto everybody.

I think we should try to change our direction: do not start teaching from the aspect of the computer and programming tools – but do start it from the human viewpoint (program is only a TOOL, not the TARGET). In this we need special elements: bricks, that resolve general, subsystem-level tasks that were drawn as real human/domain requirements, not technical, programming ones. It would be a good thing, if there should be a multi-platform coded collection of them. Not in theories, not in books, not in lectures – but in my hands.

If these "bricks" were called "patterns", let them be. There are some "classic" design patterns, that are already available in 4GL or in designer tools – but they are few and/or difficult to teach programming with them.

Paper: [WS14], Selmeci_ProgPatterns.pdf

2.7 Teaching Database Management Using Concrete Examples (Iván Seres & Ágoston Zsembery, SZÁMALK, Budapest)

In our paper, we give a short account on how we teach in our college the use an RDBMS from a Java program, and what patterns we apply for that. As a prerequisite of this course, students know, how to build user interface. We begin the course with a small example, then we widen the problem step by step, reaching a more complex and general, but still not too difficult example. We continuously give attention to the proper design and documentation techniques. At the end of the course we reach a

problem like maintaining and searching books with their authors, publishers, ISBNs and keywords; producing reference lists.

The take-off is a quite simple program: it is based on saving serializable (vector-type) objects. This program has three tiers: a graphical interface, a logic tier and a third tier for handling data objects. Our aim is to "open" this program: first to connect it to a simple database-manager (like Microsoft Access or similar), then to turn its objects to database tables, and then to transfer much of the logic into the database-manager (as stored procedures into MS SQL Server or PostgreSQL).

Paper: [WS14], Seres_Teaching.pdf

2.8 Software Engineering Apprenticeship by Immersion (Vincent Ribaud & Philippe Saliou, Université de Brest)

Inside the french university, there are two different education systems: a 5-year academic system and a 4-year technological system. At Brest university and in the field of computer science, both systems teach the same curriculum. A dedicated fifth year was designed and offered to the 4 year-system graduates. The main objectives are: mastering software engineering activities and skills, working in a team, coping with change. The plan of action is built on a 6-month team project, lead and tutored by an experienced software professional. The real-world environment is imitated as closely as possible. A contract defines the customer-supplier relationship. A real corporate baseline (by the courtesy of a software services company) sustains engineering activities and products delivery. This corporate baseline is tailored with a software development process, called 2-Track Unified Process, relying on UML models. The students are divided into two companies of 5 persons. Each company has its own office with individual working post and shares other common installations. Each company uses a different and complete software engineering tools suite (Oracle and Rational/IBM Websphere). The apprenticeship process is achieved in two iterations. During the first iteration (4 months), students are swapped around the different tasks needed by engineering activities and strongly guided by the tutor. During the second iteration (2 months), roles are fixed within each team and teams are relatively autonomous to complete the project, the tutor performing mainly a supervising and rescuing activity. The heart of the apprenticeship process is the iterations breakdown into work cards. Each work card defines the products to be delivered, the precise nature of services required, the helpful resources provided, the expected workload and planning. The assessment process is essentially formative, due to the permanent feed-back of tutoring. Nevertheless, the awarding of a diploma needs a parallel formal assessment process. The expected abilities and skills are compared with those effectively reached. In conclusion, the professional insertion and career evolution of students need to be observed over several years in order to evaluate the real benefits of this system.

Paper: [WS14], Saliou_SEAppByImmersion.pdf

2.9 Restructuration – Pedagogical Pattern (Marianna Sipos, Dennis Gabor College, Budapest)

Information Technology (IT) is in continuous change, so we need to modify our concepts and methodologies during the teaching process.

Problem: Often we face the challenge that new ideas come and go into practice within very short timescales. However, we know that teaching these new base ideas early on is essential because by reflecting the state-of-the-art of the current evolution, they influence the general thinking about the subject matter. On the other hand, there are situations when the social development or economic conditions force a new syllabus structure.

Solution: Therefore, we should use the restructuration pedagogical pattern to rearrange our syllabus, by starting the course with the new base concepts and the common materials.

We have to respect that the new base concept is the new "big idea" and thus we are teaching it as an Early Bird [ELSS]. Taking the prerequisite knowledge into account, we need to teach the new idea briefly at the beginning and come back to the subject matter as often as possible using a Spiral [BEMW] approach.

Rearranging the first part of the course has an impact on the succeeding topics. Thus we need to restructure the whole course in order to respect the new ideas.

Known Uses: Restructuration has been applied several times: when structured programming came up, or when object-oriented programming became fundamental [KV] we had to teach the object-oriented paradigm early to have time to teach the class libraries and the event driven programming. We can easily develop GUI while there are tools and the background classes support the implementations. Nowadays, distributed systems are rising to become everyday technology.

Consequences: Restructuration means not only finding a new order because a later used example can include the earlier learned material, so it is often complex. Thus, you have to find simple examples and exercises at the beginning instead of the formerly used complicated ones that came in later. **However** if you do not restructure your material you will not have enough time to teach and to delve into the more important subjects.

Related Patterns: If we might want to introduce the new base concept at an abstract level followed by concrete practical examples, as suggested by Abstraction Gravity [EMMW]. Or we might consider teaching the new ideas incrementally following Experiencing in the Tiny, Small and Large [EMMW]. The use of the Restructuration, can influence the order of the material and the essence of what you want to teach. As an Early Bird [ELSS] we can focus on distributed applications. Later we can deepen object-oriented concepts teaching such important topics as polymorphic dispatch.

Paper: [WS14], Sipos_Restructuration.pdf

2.10 Teaching Software Development in the Conversion Masters Program in the UK (Marie-Helene Ng Cheong Vee & Constantinos A. Constantinides, University of London)

In this paper we recognize that teaching software development in a conversion graduate programme needs careful organization if students are to benefit form the programme and be prepared for a career in Information Technology. We present the feedback obtained from our students in the MSc full time and part time Computer Science programme at Birkbeck College, University of London. We analysed the results we obtained from the survey we carried out towards the end of the programme and consequently suggested alternatives where applicable. We recognize the importance of being able to transfer concepts into syntax rather than the contrary. This is why choosing the right paradigm and the right language is crucial. To help students in the learning process, the use of case studies and "real world" activities such as teamwork enhance the experience and learning. Our view is that restructuring part of the programme might benefit our students even more. The idea is that students are introduced to object-oriented concepts from the start with a pure OO language and learn procedural programming as part of OO with the aim of preparing them for a successful career in software development.

Paper: [WS14], Vee_Teaching.pdf

3 Summary, Discussion

There were a lot of valuable presentations and good ideas on the workshop. It is a fact however, that nobody submitted a complete project to the workshop. The why is a question. Is it too easy? Is it too hard? Is it secret? Do anybody need to have a concrete SDP (Software Development Pack, all the documentations and source code)? Is there any on the market? The survey definitly showed that there are problems with Software Development learning support.

Let's see the summaries of participants about the workshop and the discussion:

Krzysztof Kaczmarski: Workshop participants agreed that there are no sufficient methods that can assure proper level of software development teaching. What we need is a set of easily applicable and transferable teaching patterns. There should be probably a group of interest created, which would cooperate in creating an information platform. It is quite obvious that it should use kind of web portal site with free access that would also be able to gather reviews and focus interest.

Dénes Medzihradszky: During the Workshop the pattern definition was interpreted in its fullest meaning from teaching patterns to software development patterns. It was interesting to note, that real patterns are not frequently used in teaching, although everybody agreed that patterns are providing valuable help in the development work. Practically all the lectures about teaching courses in software development stated, that full documentation is required along the software (these together would make excellent patterns!) but no patterns were provided before the develop-

ment work. During the discussion it became clear, that the main problem is the lack of suitable patterns. We all agreed, that it would be a great step forward to provide those for our students. Most of the speakers attending the discussion were rather enthusiastic about contributing material for a planned SDP Internet site (Software Development Packages/Patterns). The main reason behind the lack of available pattern examples was agreed to be the lack of time and resources.

Marie-Helene Ng Cheong Vee & Constantinos A. Constantinides: The various papers presented at the workshop showed alternative approaches and tools used for teaching software development at various institutions. These approaches were assessed and problems encountered discussed. It was generally agreed that generic use cases and re-usable teaching patterns for software development are missing. These might be useful in saving time and effort in education. But it is also recognized that deriving such patterns and use-cases are very difficult to achieve.

Thomas Cleenewerck: The workshops main focus was on teaching software development (SD). It is very hard to come up with more large scale examples of SD because it is very hard to put the essence of SD in these examples. The essence in teaching software development is to teach students to come up with a design and implementation that (1) must be fit for their or a general interpretation of the intended software; (2) consists of third-party components; (3) can easily evolve and change over time. These boil down to the fundamental design issues of low-coupling, high cohesion, managing the interactions, etc. They can be far more easily put in smaller examples, and explained in a manageable environment like the one presented in the lost in object-space paper.

Iván Seres: We all agreed that using patterns are very important in software development. Perhaps, the main problem is that important patterns, applicable for real, tangible and essential issues are owed by software development firms, and that it is not in these enterprises' interest to publish them (at any rate, not in the short run). Nevertheless, students, learning software development, teachers, tutors and for those who develop their software products in a small business it would be very good to rely on a good and rich repository of usable, detailed patterns. Moreover, concrete examples to patterns is substantial, as there are many software developers of non-mathematical qualification, and the abstract ideas are not ideal tools to fascinate their brains. In a talk after the presentation has been outlined an idea about establishing a web site; maybe that web site with its competitions, maturation process, awards and general reputation would become a "condensation point" for making and publishing complete software development documentations. I would take part willingly in such a task.

Vincent Ribaud & Philippe Saliou: The workshop contributions presented different ways or pedagogical choices of teaching software development. A software project is the support most commonly used. In a certain sense, the software project emphasizes the activities (essentially programming and testing) that students are able to carry out without help. By that very fact, a project requires less teacher involvment in comparison with the classical way of teaching. In the main, students accomplish the learning process by comparing the expected work, the guidelines and the examples. So, we agree with the essential conclusion of the workshop, i.e. providing small but

complete development examples. Complete examples mean that all the intermediary products of the development cycle are available.

Jürgen Börstler: Patterns by themselves are not "good enough" for the classroom. The average student (and lecturer as well) needs actual examples of contexts where the application of certain patterns could lead to improvements. The improvements should furthermore be carefully analyzed and documented. An area that seems to be specifically neglected is software development as a whole. There are very few examples of (team) projects that are fully documented, i.e. from project proposal to running code (see for example [UPEDU]). But the problem is worse than that. To provide insight into the development process one would not only need the results. The most interesting parts are the decisions and analyses that lead to those results. Workshop participants agreed that patterns, or software development examples in general, need to be accompanied by instructional material. A repository of such materials would be very beneficial.

Marianna Sipos: The workshop was about where and how can we use patterns in teaching software development. The quality of our teaching depends on our knowledge and on our teaching methods. Thus I think there are two kinds of patterns which we can use, software development patterns and pedagogical patterns.

The first software development pattern book Design Patterns [GHJV], which is well known all over the world was published in 1995. From this time large numbers of pattern books have been published and at the pattern conferences every year large number of patterns are explored, developed and celebrated on every continent. The patterns relate to software development are used in companies and learned on universities.

Human interaction in teaching and co-operation is also one focus of PLoP conferences, but I do not know any book about Pedagogical Patterns.

The workshop discussed the use of patterns, and the problems in teaching software development. We could see some interesting and useful techniques how try teachers give knowledge and key qualifications for students in different countries.

Erzsébet Angster: Workshop participants were agreed that developing concrete, „real-world" projects are very important in teaching SD. We also agreed that there are not enough practical teaching materials, and there are not enough concrete and complete SD examples. The survey about the Software Development Learning Support underlined this. A website with complete projects (documentation, source and teaching materials) would help the teachers.

Joe Bergin: The reader probably understands, correctly, that there were a wide variety of ideas presented in this workshop and not all participants were in agreement on the most important things. This points to the richness of the field and also to the fact that it is not yet mature and that we are all struggling with how to best teach software development. There was wide agreement, however, that the original idea of E. Angster to develop a repository of rich examples of complete interesting systems is a good one and should be pursued. We hope that the participants here and others in the community will join in this effort.

References

[GHJV] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns, Addison Wesley Longman Inc. 1995

[POSA] Buschmann, Frank (1996) – Meunier, Regine – Rohnert, Hans – Sommerlad, Peter – STAL, Michael: Pattern-Oriented Software Architecture. A System of Patterns. Chichester: John Wiley & Sons, 1996

[UPEDU] Unified Process for EDUcation, http://www.upedu.org/upedu

[BEMW] Joseph Bergin, Jutta Eckstein, Mary Lynn Manns, Eugene Wallingford. *Patterns for Gaining Different Perspectives,* Proceedings of PLoP 2001.

[ELSS] Jutta Eckstein, Mary Lynn Manns, Helen Sharp, Marianna Sipos, Patterns for Teaching Effectively, EuroPLoP 2003.

[EMMW] Jutta Eckstein, Mary Lynn Manns, Eugene Wallingford: Patterns for Experiential Learning, Proceedings of EuroPLoP 2001.

[KV] Kerstin Voigt: Big Picture on a Small Scale (BPSS) Pattern, http://www-lifia.info.edu.ar/ppp/pp46

[WS14] Workshop website: http://www.gdf.hu/progtanszek/angster/ecoop03

Object-Oriented Language Engineering for the Post-Java Era

Wolfgang De Meuter[1], Stéphane Ducasse[2], Theo D'Hondt[1], and Ole-Lehrman Madsen[3]

[1] Programming Technology Lab - Computer Science Department, Faculty of Sciences, Vrije Universiteit Brusse, Belgium
[2] Software Composition Group, University of Berne, Switzerland
[3] Department of Computer Science, Aarhus University, Denmark

Abstract. This report covers the activities of the workshop on "Object-Oriented Language Engineering for the Post-Java Era". We describe the context that led to submission of the workshop proposal, the organisation of the workshop and the results presented at the workshop.

1 Introduction

Since the advent of Java, object-oriented language design clearly does not fill up proceedings of ECOOP's and OOPSLA's anymore the way it used to. Nevertheless, we have the general feeling that many researchers in the object-oriented community are still interested in, and still feel the need for research in object-oriented language design. Proof of this are a series of language related events, in the near past, that were all able to attract a huge number of participants:

- At Markku Sakkinens Inheritance Workshop at ECOOP02 in Malaga, Spain, more than 30 researchers were attracted by the central question of how inheritance has evolved in the past few years.
- Richard Gabriel's Feyerabend events at past OOPSLAs and ECOOPs attracted a huge number of participants. Many of them were distinguished Lisp or Smalltalk people, and/or distinguished language designers.
- At the Onward! track at OOPSLA, a.o., new programming paradigms are the topic of study. The fact that this succesful track is part of a conference like OOPSLA shows the connection with object-orientation.
- Last but not least, during several (both formal and informal) speeches, important guys in the field keep on repeating that Java maybe a means, but certainly is not the end.

In order to verify these "general feelings that (a) less and less resources are being spent on language design" and that (b) "almost all language design in OO currently is formulated as Java extensions", we did a quantitative survey of past ECOOP and OOPSLA proceedings. The outcome is listed in the following table:

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 143–153, 2004.

	ECOOP				OOPSLA			
1986					50	28	14	0
1987	25	20	5	0	44	13,64	6	0
1988	23	17,39	4	0	30	6,67	2	0
1989	21	23,81	5	0	45	28,89	13	0
1990	29	34,48	10	0	29	34,48	10	0
1991	22	22,73	5	0	23	17,39	4	0
1992	23	26,09	6	0	31	29,03	9	0
1993	24	25	6	0	26	11,54	3	0
1994	25	24	6	0	41	17,07	7	0
1995	18	22,22	4	0	29	13,79	4	0
1996	21	38,1	8	0	26	19,23	5	0
1997	20	50	8	2	21	14,29	2	1
1998	24	25	3	3	27	22,22	3	3
1999	20	25	3	2	30	10	3	0
2000	20	35	1	6	26	19,23	0	5
2001	18	27,78	3	2	27	22,23	0	6
2002	24	29,17	1	6	25	16	0	4

The table lists both ECOOP and OOPSLA figures per year. The first column for each conference shows the number of papers presented. The second column shows the percentage of papers on language design (including those about Java extensions). The third column the number of papers on language design which are not about Java and the final column shows the number of language design papers that are formulated as a Java extension. The table confirms our feelings: since the widespread acceptation of Java, the number of papers on "pure language design" decreases dramatically and the research area of object-oriented language design is gradually taken over by Java extension papers. We can see that the situation is more extreme at OOPSLA than at ECOOP. The point of departure of this workshop is that this is a deplorable situation because we believe that Java is not the end of object-orientation and because we believe that in a number of emerging fields (such as components and mobility), Java is *not* the way to go. It was this context that gave rise to submitting the workshop proposal to the ECOOP03 workshop committee. Our formulation was as follows:

The advent of Java has always been perceived as a major breakthrough in the realm of object-oriented languages. And to some extent it was: it turned academic features like interfaces, garbage-collection and meta-programming into technologies generally accepted by industry. Nevertheless Java also acted as a brake especially to academic language design research. Whereas pre-Java Ecoops and Oopslas traditionally featured several tracks with a plethora of research results in language design, more recent versions of these conferences show far less of these. And those results that do make it to the proceedings very often are formulated as extensions of Java. Therefore they necessarily follow the Java-doctrine: statically typed single-inheritance class-based languages with interfaces and exception handling. On the grapevine we know that people are still interested in language design that radically diverges from this doctrine. Their papers some-

how don't seem to make it because of several reasons. The goal of this workshop was to bring together researchers in object oriented language design who adhere language features and languages that do not fit into the mainstream.

In the call for position papers the listed topics of interested was formulated as follows:

- agent languages
- distributed languages
- actors, active objects
- mixins
- prototypes
- multi-paradigm
- reflection and meta-programming
- ... all other exotic language features which you would categorize as OO

Attendance

In order to attend the workshop, one was expected to submit a long (max 10 pages) paper presenting scientific results about OO language design (a language, a feature, ...) or a short (max 5 pages) essay defending a position about whereto object-oriented language design should be heading. We received 14 submissions, most of them long papers. Of those 14 submissions, 13 authors attended the workshop. But apart from these, we had a lot of attendees that asked us whether it was possible to participate in the workshop without a position paper. This resulted in 42 participants! We believe this huge number of participants confirms our conjecture that people still find object-oriented language engineering useful and interesting!

2 Organization

We did not want to have a "mini conference" workshop with paper presentations all day long. On the other hand, we did not want to have unstructured discussions without pre-planned activities either. We therefore pursued a middle course and divided the day into a paper session and a discussion session:

- Before lunch, a number (7) of selected position papers were presented. Instead of selecting the papers ourselves, we organized a voting over email in the weeks preceding the workshop: everyone that submitted a position paper was allowed to vote over which paper he or she would like to have been presented. The 6 papers with the highest score were selected for presentation. Apart from those, a 7th position paper (by Andrew Black) was selected by us because of its visionariness. In what follows, we list the presentations that filled the morning session of the workshop (see section 3 for an overview of the content of the papers and for a complete list of authors).
  - The successes and Failures of a Language as a Language Extension (presentation by Kasper Graversen)

- Dynamically Scoped Functions as the Essence of AOP (presentation Pascal Costanza)
- Power to Collections! (presentation by Philippe Mougin)
- Wild Abstraction Ideas for Highly Dynamic Software. (presentation by Wolfgang De Meuter)
- Orthogonality in Language Design - Why and How to Fake it. (presentation by Stephan Herrmann)
- Language Support for Multi-Paradigm (presentation by Matthias Hoelzl)
- **Post Javaism** (invited presentation by Andrew Black)

– After lunch, we filled the afternoon with a plenary discussion. In order to keep the discussion more or less structured and meaningful, we asked a number of "distinguished language designers in the audience" to sit upfront and act as an informal panel by which the discussions could be guided. The panel acted as a lively source of knowledge and as the historical conscience of the workshop. Although the panel had quite some "names" this did not act as a drag: many participants actively took part of the discussions. The overal result was a plenary discussion in which the panel acted as a mirror to reflect ideas on.

  The members of the panel were: Ole-Lehrman Madsen, Markku Sakkinnen, Kim Bruce, Andrew Black, Theo D'Hondt and Gilad Bracha.

  Right before the lunch break we had a list circulated on which participants could write questions to be asked to the panel in case the discussion stalked. The collected questions were:

  - What is a good exception mechanism for OO?
  - Political Question: What can we do to make the masses consider non-mainstream languages again?
  - Should AOP be defined in terms of reflection or should we continue to hide the meta in AOP?
  - How far should a language go in being reflective?
  - Is Java a blind ally? That is: is it possible to go *o*n from Java or should we go *b*ack?
  - Is C# a post-Java era language or a pre-Java language? (this question was unanimously answered as "C# *is* a Java-era language!")
  - Is Java really the most modern OO language? Isn't Beta much more advanced than Java?
  - Is Java-baching at all going to help in evolving to a "better" language?
  - General Question: Where will be be (or should we be) in 3 to 5 years from now?
  - Do we need static type checking? If yes, what should the type system be able to ensure?
  - Is a ML-like (inferred!) type system possible for OO? Is it a dream?
  - Are Ruby and Python post-Java enough?
  - What are the new markets for future languages?

  Some of these questions where effectively used to feed the discussion. Others were discarded by the panelists.

3 Submitted Position Papers

3.1 Synchronization with Type Variables

Synchronization with Type Variables by Franz Puntigam
(franz@a0.complang.tuwien.ac.at)

Process types allow programmers to specify constraints on acceptable method calls [8,9]. For example, two kinds of calls shall be accepted only in alternation. A compiler can statically ensure that all calls in a system are acceptable even if there exist any number of references to an object. The type concept supports subtyping, parametric polymorphism, and separate compilation. These properties make process types useful as a basis for the specification of reliable communication in sequential as well as concurrent systems. Previous work is improved mainly by considering (1) variables that belong to types and (2) spontaneous modifications of these variables usable for the dynamic synchronization of concurrent processes.

3.2 Test Composition with Example Objects and Example Methods

By Markus Gälli
(markus.gaelli@iam.unibe.ch).

While assertions of *Design by Contract* from Eiffel [10] found its way into the language-definitions of Python and of Java SDK 1.4, current object-oriented languages do not make the concepts of *unit-testing* [11] explicit in their definitions or meta-models.

Not having support of unit-testing in a programming language makes it harder to compose and re-compose [12] test-scenarios and tests.

We propose, that an object-oriented language should include explicit concepts for example objects and example methods. This concepts ease the composition of complex test-scenarios, they help to refactor the program with the tests and also to keep the duration of the tests as low and the coverage of the tests as high as possible.

3.3 Wild Abstraction Ideas for Highly Dynamic Software

By Wolfgang De Meuter, Jessie Dedecker and Theo D'Hondt
(wdmeuter@vub.ac.be).

In this position statement, the authors propose a few roughly sketched language features of which they feel that they might dramatically change the way we can structure and reason about systems that have to operate in highly flexible environments, such as moving PDA's. The features proposed have never been incorporated in a real language and have never been implemented. The goal was to be provocative rather than demonstrative. Of the features discussed, the most innovative ones are multivalues (values that can *be* several objects at the same time), distributed inheritance (prototype-based objects of which the parts reside on different machines) and cloning families (properties shared by families of copies that are automatically kept consistent).

3.4 Making Safer Object-Oriented Languages

By Krzysztof Kaczmarski.

In widely used programming languages and in databases there is often a need of assigning no value to a variable. Most systems often use undefined, unknown, nil or null to indicate that some value is unknown. However, semantics of these notions is rather difficult. In an object-oriented paradigm missing information as simply missing attributes of objects has evolved to semi-structured data and is under research now. Modern imperative programming language should extend the control of missing information. Optional attributes should be distinguished from obligatory ones. Each usage of an possibly missing value should be guarded by special constructs, forcing a programmer to expect empty data. On the other hand in certain places compiler and programmer could be also sure that is dealing with existing values. Most important expected profits of such new syntax and semantics are more reliable applications and simplified code provers and verifiers.

3.5 Hosting Object-Oriented Programming in Totally Functional Programming with Characteristic Methods

By Paul Bailes, Colin Kemp and Sean Seefried. A functional representation for objects seems to offer an inherent cohesiveness for class methods. Functional representations for data are indicated by the affinity between programming and language extension: if language extension should exclude interpretation of symbolic data in favour of direct definition of functions, then in programming it should be preferable to discover and use directly through functional representations the applicative behaviours inherent in symbolic data. This approach is echoed in a fundamental affinity between object-oriented and functional programming, but the idea that every object has one characteristic applicative behaviour needs to be reconciled with the tradition of multiple methods in OOP. The higher-order methods to which we have recourse not only show promise in affecting this reconciliation, but also have potential to measure the semantic cohesion present in a multi-method class.

3.6 The Successes and Failures of a Language as a Language Extension

By Kasper B. Graversen
(`kbilsted@it-c.dk`).

This paper is an experience report on implementing the Chameleon role model as a language extension to Java. First, we introduce our role model on a conceptual level, then an overview of the shortcomings of Java. The shortcomings are interesting in that they shed light on the degree of control needed in order to reuse the language to be extended. In the contemplation parts of the paper, we advocate research that tries to abstract concepts to the extend possible, as seen with the pattern concept in Beta. And we advocate language design that permits recursive use of its concept, such as in our role model, where roles can be roles

for objects as well as roles. Finally we identify, that language implementations are about two things. 1) The ability to elegantly express the semantics of the languageÑbe it as a set of transformations of an existing language or a set of assembly instructions. And 2) to provide access to a large collection of common functionality, exemplified in this paper by the Java API, in order for the language to be used in real-life scenarios.

3.7 Orthogonality in Language Design D – Why and How to Fake It

By Stephan Herrmann
(stephan@cs.tu-berlin.de).

Using examples of oddities in Java, it is argued that language features in isolation are less a problem than integrating them into a programming language. For this hard problem a method is still lacking. Experience from developing the language ObjectTeams/Java is reported as a basis for discussing orthogonality of language features. Object Teams solve a large number of modularity issues by means of a minimal set of new features. Key concepts are Team-modules containing roles that can be bound to existing classes. While those features can't be truly orthogonal as demonstrated by the example of Team activation and exception handling, composability can yet be achieved by clever translation such that programmers may work with a sound mental model in which features are independent [7], [3], and[5].

3.8 Linguistic Symbiosis Through Coroutined Interpretation

By Theo D'Hondt, Kris Gybels, Maja D'Hondt and Adriaan Peeters
(tjdhondt@vub.ac.be).

In the world of computer languages linguistic symbiosis describes the process whereby multiple programming styles or paradigms are instantiated in one composite language. In some cases symbiosis is natural because one particular paradigm can easily be expressed on top of another; in others it is easy to construct because the cohabitation of some paradigms is obvious. Languages like Scheme merge a functional style with an imperative one without too much hassle; in a similar vein C++ merges objects into a procedural language. The object-oriented paradigm seems to be compatible with most others; the concurrent programming paradigm less so; and the logic programming paradigm seems to be most resistant to symbiosis, although logic can be eminently used to formulate other paradigms. We are interested in true symbiosis: a context that allows expressions to be formulated in all of the participating paradigms, without preferential treatment of one or the other. In this paper we propose to use coroutined interpretation to support such a symbiosis.

3.9 Language Support for Multi-Paradigm Programming

By Matthias M. Holzl
(hoelzl@informatik.uni-muenchen.de). To build better programs we have to

reduce the complexity of software development. We argue that this requires languages that integrate different programming paradigms and provide means for metalinguistic abstraction. But language features alone are not sufficient, it is equally important to provide a development environment that faciliates the task of software development, e.g., by providing a Socratic reasoner, declarative information about the program, or architecture information. Further readings are Hoelzl:2001,Hoelzl:2002a,Hoelzl:2002b.

3.10 Dynamically Scoped Functions as the Essence of AOP

By Pascal Costanza
(costanza@web.de).

This paper demonstrates that a focus on a strongly dynamically-natured programming language constructs reveals a very simple mechanism behind the notions of aspect-oriented programming. Whereas the AOP community usually tries to tackle these notions with complex static means, that paper shows how plain dynamic scoping for functions together with a structured way to call the previous definition of a function provides a full-fledged aspect-oriented approach. Furthermore, the appendix of that paper includes a complete executable one-page implementation of this idea. This is possible because the sufficiently complete and well-balanced language Common Lisp is used. A slightly extended version of that paper is published in SIGPLAN Notices 38(8), August 2003.

3.11 Open Surfaces for Controlled Visibility

By Stéphane Ducasse, Nathanael Schaerli and Roel Wuyts
(ducasse@iam.unibe.ch).

Current languages contain visibility mechanisms such as *private*, *protected*, or *public* to control who can see what. However, these visibility mechanisms are *fixed once for all*. Moreover, they do not solve all problems related to the visibility, and are typically of a static nature. This position paper [1] presents an open and uniform way of dealing with visibility and introduce *Surfaces*: *i.e.,* list of methods that control the way the behavior of an object is accessible. Surfaces provide a mechanism that allow a class to expose groups of methods under a certain name. Moreover, the mechanism is open so that new surfaces can be added. This allows a class to give methods certain visibility for one client, and another visibility for other clients. Note also that different instances of the same class could see the same object with different surfaces, since the surfaces are not applied on classes but on objects. Moreover, surfaces support parametrised usage, where the client can choose between different options that are offered. There is currently no implementation that supports surfaces, but we have a very good idea on how to add surfaces to the Squeak Smalltalk system [2] using techniques analogue to the implementation of the Traits system [6].

3.12 The ClassBox Module System

By Alexandre Bergel, Stéphane Ducasse and Roel Wuyts
(bergel@iam.unibe.ch).

The Classbox Model is an extension to the OO paradigm bringing in a powerful modular development. It allows to reuse and extend some code under a modular unit, named Classbox. A Classbox is an unit of scoping composed by some class imports declarations and by definitions of class and method.

Only classes can be imported. The granularity of the import clause can be unitary (a classbox can import classes one by one) or by whole Classbox (a classbox can import all the classes defined by an another classbox). The scope of a classbox is the classbox itself. This mean a classbox can add or redefine any method of a class without breaking any client of the imported classes.

One strength of the model is a class extension offered by a classbox is not simply a new layer put in front of the formally code: The formally and new code can interact each other without any restriction. A validation is made in Squeak using a modified virtual machine. In order to have a complete model and some acceptable performance the method lookup need to take the scope into account.

3.13 Power to Collections! Generalizing Polymorphism by Unifying Array Programming and Object-Oriented Programming

By Stéphane Ducasse and Philippe Mougin
(ducasse@iam.unibe.ch).

Array programming shines in its ability to express computations at a high-level of abstraction, allowing one to manipulate and query whole *sets* of data at *once*. This paper presents the OOPAL model that enhances object-oriented programming with array programming features. The goal of OOPAL is to determine a minimum set of modifications that must be made to the traditional object model in order to take advantage of the possibilities of array programming. It is based on a minimal extension of method invocation and the definition of a kernel of methods implementing the fundamental array programming operations. The model is validated in F-Script, a new scripting language. A long version of this paper is [4] to which the interested reader can find implementation notes.

4 "Post Javaism" by Andrew P. Black

This position paper was selected by the organisers to be presented as a "long presentation" right before the lunch break. It was considered to be thought-provoking. We summarize the paper here.

Summary of the Paper

Professor Andrew Black concluded the morning session with a presentation that summarized the relationship between genres of architecture and programming

language. Despite the enticing similarity of their names, the correspondence is not between post-modern architecture and post-Java language design. Instead, modernism in architecture, which discarded accepted but outmoded conventions and was characterized by the slogans "form follows function" and "less is more", corresponds quite closely to Smalltalk, which likewise discarded accepted conventions of syntax and semantics and replaced them by a highly functional but extremely spare language kernel.

Post-modern architecture started from the observation that many people were uncomfortable with the stark reality of modernism, and seemed to feel more comfortable with classical forms, even if they were no longer strictly necessary. It proceeded to incorporate illusions to bygone architectural fashions, sometimes in whimsical ways. Similarly, the Java language incorporated many structural elements that were not strictly required by the object-oriented style, but which alluded back to C and C++ and thus made programmers feel more comfortable at the expense of added complexity.

The successors to the post-modernists, notably the architect Renzo Piano, have discarded many of post-modernism's excrescences and embellishments and returned to a more people-friendly form of modernism in which new technologies are used to create a building that responds to its context. Black argued that we might expect to see a similar trend in post-Java language design: a return to the more streamlined forms of Smalltalk suitably modified to meet the needs of the Internet age, and taking advantage of new technologies such as type inference. Specifically, he suggested that Smalltalk's success could be attributable to its small size, its absence of explicit type declarations, and the excellence of its programming environment. Its preference for higher-order features such as closures (blocks) rather than special "features" such as exception handling has contributed to its remarkable longevity. A successor to Java would do well to imitate these strengths, and complement them with a declarative language syntax, array expressions, namespaces, explicit interfaces coupled with implicit conformance between interface and implementation, and more disciplined mechanisms for reflection and initialization. Blacks presentation was followed by a lively discussion which continued into lunch.

References

1. Stéphane Ducasse, Nathanael Schaerli, and Roel Wuyts. Controlled right accesses based on uniform and open surfaces. In *Proceedings of the ECOOP '03 Workshop on Object-oriented Language Engineering for the Post-Java Era*, July 2003.
2. Mark Guzdial. *Squeak – Object Oriented Design with Multimedia Applications.* Prentice-Hall, 2001.
3. Stephan Herrmann. Object teams: Improving modularity for crosscutting collaborations. In *Proc. Net Object Days 2002*, volume 2591 of *Lecture Notes in Computer Science*, pages 248–264. Springer, 2003.
4. Philippe Mougin and Stéphane Ducasse. Oopal: Integrating array programming in object-oriented programming. In *OOPSLA'2003 (International Conference on Object-Oriented Programming Systems, Languages and Applications)*, 2003. 26 papiers acceptés sur 142, 26 accepted papers on 142 = 18%.

5. Object Teams home page. http://www.ObjectTeams.org.
6. Nathanael Schärli, Stéphane Ducasse, Oscar Nierstrasz, and Andrew Black. Traits: Composable units of behavior. In *Proceedings ECOOP 2003*, LNCS, pages 248–274. Springer Verlag, jul 2003.
7. M. Veit and S. Herrmann. Model-view-controller and object teams: A perfect match of paradigms. In *Proc. of 2nd International Conference on Aspect Oriented Software Development*, pages 140–149, Boston, USA, March 2003. ACM Press.
8. Franz Puntigam. Coordination Requirements Expressed in Types for Active Objects. In *Proceedings ECOOP'97*, LNCS 1241, pages 367–388, Jyväskylä, Finland, June 1997. Spriner.
9. Franz Puntigam. *Concurrent Object-Oriented Programming with Process Types.* Der Andere Verlag, 2000.
10. Bertrand Meyer. *Object-Oriented Software Construction (2nd edition).* Prentice-Hall, 1997.
11. Kent Beck. *Test Driven Development: By Example.* Addison-Wesley, 2003.
12. Arie van Deursen, Leon Moonen, Alex ban den Bergh, Gerar Kok. Refactoring Test Code. In *Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes (XP2001)*, pages 92–95, 2001.

Analysis of Aspect-Oriented Software

Jan Hannemann[1], Ruzanna Chitchyan[2], and Awais Rashid[2]

[1] Department of Computer Science, University of British Columbia, Canada,
jan@cs.ubc.ca
[2] Computing Department, Lancaster University, UK,
r.chitchyan@lancaster.ac.uk, awais@comp.lancs.ac.uk

Abstract. This report summarizes the various discussions and results from the Workshop on Analysis of Aspect-Oriented Software held in conjunction with the European Conference on Object-Oriented Programming (ECOOP) in Darmstadt, Germany, July 2003.

1 Introduction

Aspect-oriented software development (AOSD) is a new approach for modularizing complex software systems. As an extension to other software development paradigms, such as object-oriented (OO) development, it allows to capture and modularize concerns that crosscut a software system in so-called aspects. Aspects are constructs that can be identified and manipulated throughout the development process.

Much work has been done to develop aspect-oriented (AO) languages, support, and models for AOSD. With the growing acceptance of AOSD as a software development technique, dedicated support for analysis of aspectual artifacts is needed at the various stages of software development. Developers need to be able to understand, visualize, specify, verify and test aspect-oriented requirements, architectures, designs and programs to make it an industrially viable technology.

This report aims at providing an overview of various issues pertaining to analysis of AO software: where such analysis is needed, what the key challenges are and how these challenges can be addressed. These questions must be considered in the context of the whole software development life cycle. The report is organized as follows: Section 2 provides an overview of current research in this area, based on paper submissions to this workshop. Section 3 discusses whether AOSD improves the ability to reason about the correctness of a system. The discussion is based on a panel held at the workshop. Section 4 focuses on three key topics (based on the group discussions at the workshop): adapting present analysis techniques to an AO context, how conflicts and inconsistencies in AO software may be identified and resolved and how impact analysis of AO software may be undertaken. Section 5 summarizes the most important findings of the workshop and outlines potential future directions for research on analysis of AO software.

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 154–164, 2004.

2 Current Research

The various papers at the workshop [6] focused on three facets of analysis of AO software: requirements and design analysis, tool support and comparative studies of AO and non-AO implementations. This section summarizes the contributions.

2.1 Requirements and Design Analysis

Constantinides [3] advocates the need to treat aspects as first class entities throughout the software lifecycle. Through a case study, the author demonstrates how crosscutting concerns become visible during requirements analysis in the pre- and post-conditions of system use cases and constraints on sequence diagrams and how they may be mapped to classes at the design stage and subsequently to implementation modules. Hachani and Bardou [5] focus on the implications of implementing OO design patterns with AO languages. Through an implementation of the *Strategy* pattern they argue that application of AO to these patterns should be visible not only at the implementation level but also at design pattern description level. Bertagnolli and Lisboa [4] focus on a model for encapsulating non-functional requirements in aspects, in order to improve analyzability of the functional requirements and components.

2.2 Tool Support

Clement, Colyer and Kersten [2] discuss current support and future extensions to AspectJ Development Tools (AJDT) to support analysis and understanding of AspectJ programs. In this context, they focus on the current support for annotations, visualization and debugging and future extensions in the form of tool-tips, content assistance, a Pointcut Wizard and Pointcut Reader. Bergmans [1] observes that aspects affect the behavior of the base program code and hence, may introduce conflicts into the system. The author identifies several properties that affect the ability to reason about AO software including, obliviousness, open-ended specifications of joinpoints, lack of semantic analyzability and interactions among aspects. The author also outlines a semantic conflict checker tool, to be based on the Composition Filters model. Stoerzer, Krinke and Breu [9] propose the use of program traces as a means for analyzing the changed behavior of a program due to aspect weaving. Trace difference analysis – comparing traces of the program version before and after aspect composition – is used to reveal the precise points and causes of change introduced by aspect into the base program.

2.3 Comparison of AO and Non-AO Implementations

Slowikowski and Zielinski [8] provide examples of security concerns in component-based applications implemented as container-managed features and as AspectJ aspects. They suggest that the two solutions could be complementary, and could be used in combination. Reina, Torres and Toro [7], on the other

hand, compare separation of concerns in AspectJ with that in CACOON. They observe that while an AO solution provides better localization, the XML–based solution allows for better expressiveness.

3 Panel: Does AOSD Improve the Ability to Reason About the Correctness of a System?

Panel chair: *Awais Rashid (Lancaster University, UK)*

Panelists: *Adrian Coyler (IBM Hursley Labs, UK), Lodewijk Bergmans (University of Twente, The Netherlands), Klaus Ostermann (Technical University of Darmstadt, Germany), Elke Pulvermüller (University of Karlsruhe, Germany)*

This section summarizes the panel discussion that took place at the workshop[1]. The aim of the panel was to discuss whether AOSD improves or impedes the ability to reason about the ccorrectness of a software system (this should not be confused with formal proofs of program correctness). *Rashid* opened the discussion by contrasting two different views of AOSD and the ability to reason about correctness. Using requirements analysis as a basis, he observed that AOSD improves the ability to reason about correctness, as one can understand the influence of an aspectual requirement on non-aspectual requirements. However, at the same time, if multiple aspectual requirements influence the same set of non-aspectual requirements, the resulting trade-offs and interaction can reduce this ability. This contrast of views was also present in the panelists' position statements.

Colyer argued that larger software systems, probably developed by a team and may be through several releases, need to be reasoned about in the same way as we do for all complex systems – by applying a divide and conquer strategy to understand the parts of the system in isolation, and then putting the pieces together to form a whole. He observed that AOSD supports this line of reasoning fully. First the main path logic can be understood in isolation, free from consideration of other concerns (that is to say, tangling introduces cognitive overhead and distracts from the main flow of concentration). Then, each supplemental concern can be studied and understood in isolation. AOSD is of tremendous benefit in this case because an aspect can state clearly not only what behavior it implements, but also where that behavior should apply. The switch from a scattered implementation to an explicit statement of policy makes the structure significantly easier to reason about. When it comes to putting the parts together to form a whole, the aspect declarations provide a succinct and clear specification of how the pieces should integrate. He also highlighted the role of tool support in this context.

[1] The full statements from the panelists can be found on the workshop Web site [6] .

Bergmans, in contrast, observed that the introduction of new aspects might cause certain modules to cease functioning as expected, due to conflicts, or interference, between the superimposed aspects. This is particularly inconvenient, since the occurrence of the problems may be unpredictable to the responsible developer(s): these aspects can be introduced independently, also at different times, and the programmer of each aspect may be unaware of the existence or future introduction of the other aspect(s). He argued that one of the general difficulties that AO brings, which makes it harder to reason about the correctness of systems, is that no definite correctness observations about a particular piece of software can be made without full global knowledge (i.e. the assurance that no aspect superimposes some behavior/advice on an element that breaks down its correctness).

Ostermann observed that in contrast to the improved locality and modularity of the individual concerns, AOSD languages tend to destroy the locality of the run-time control flow – one can no longer deduce the flow of control in a module and its interaction with other modules by looking at the module only. Though tools such as AJDT use static analysis in order to infer the interactions between the modules, such analysis is only possible for very static, syntax-directed join point languages. He argued that the problem is that current join point languages are too low-level – they don't allow us to state our intention directly, but instead we have to encode it more or less as expressions over the syntax tree. For the next generation of AO languages, he envisioned much more dynamic, powerful join point languages – hence static analysis not being an option in the long run.

Pulvermueller was of the view that AOSD and system correctness are orthogonal to each other and, therefore, AOSD does not per-se improve correctness. She argued that AOSD improves the flexibility to express a program by allowing invasive changes at more or less limited program locations. It also improves the potential for modularity and flexible system configuration. These two issues (invasive changes and modularity), on the other hand, bear a challenge for correctness-ensuring techniques. We, therefore, have additional challenges in reasoning about the correctness of AOSD systems. Therefore, we need appropriate models for aspects and aspect systems abstracting from the concrete system and allowing verification of specific properties. Basic techniques for this task are already available in the form of the body of work on correctness of model-based software composition and ready to be used in AOSD. In her opinion, the primary goal, therefore, should be not to reinvent correctness-ensuring techniques but first exploit existing ones.

The panel concluded that while the additional modularisation support offered by AOSD makes it possible to reason about broadly scoped concerns, at the same time there are global properties of the system that must be reasoned about in a holistic fashion. Furthermore, development of reasoning frameworks for the purpose should draw upon established reasoning and correctness techniques and consider how mechanisms such as intentional pointcuts might change the way we reason about the correctness of AOP systems.

4 Three Key Research Questions in Analysis of AO Software and Potential Solutions

Three key issues related to the analysis of aspect-oriented software were discussed in small group settings. Each group focused on identifying the challenges and/or problems and providing potential solutions whenever possible. In the following, we look at each of the issues discussed in more detail. Please note that we differentiate here between base concerns and aspects, to allow for a clearer description of causes and effects. This is not to imply that aspects are not part of the base or are to be treated differently from other parts of the system.

4.1 Adapting Present Analysis Techniques to Aspect-Oriented Contexts

Discussion members: *Marie Beurton-Aimar (LaBRI, Université Bordeaux, France), Silvia Breu (Universität Passau, Germany), Constantinos Constantinides (University of London, UK), Thomas Cottenier (Illinois Institute of Technology, USA), Sergio Soares (Universidade Federal de Pernambuco, Brazil), Daniel Speicher (University of Bonn, Germany)*

Motivation: Although AOSD is a new software development paradigm, it builds on the strengths of traditional paradigms and provides additional means for addressing their weaknesses. Consequently, it should be possible to adapt accepted and tested development techniques, tailored for traditional paradigms, to AO concepts. We need to accommodate AOSD into the software development processes and extend traditional design techniques to accommodate AOSD.

Incorporating AOSD into Traditional Software Process Models: AOSD would align well with iterative software engineering processes. This is particularly true for the initial stages of adoption of AOSD, when there are no (or only limited) trusted ways of identifying *all* relevant crosscutting concerns, and so additional crosscutting concerns could be identified at any stage of software development. If such an additional concern is identified, the non-iterative software development process will require re-engineering or re-factoring of code and change propagation through all design and analysis artifacts. The iterative process, on the other hand, will allow incorporation of the new concern at the next iteration cycle.

Extending Traditional Design Techniques: In order to facilitate adoption of AOSD with the presently dominant software development paradigm (OO), we can augment the UML with respective features. This may be done by extending existing UML artifacts, or by introducing new ones. One of the questions raised by the introduction of aspects is whether an aspect should be allowed to change

the semantics of a particular artifact that it crosscuts (e.g. a class) or only that of the overall system as a black box, perhaps by adding behavior.

Thus, vital to the UML support for AOSD are the notion of composition between the core functionality and crosscutting concerns (including modeling support for such concepts as pointcuts, advice, etc.), and the ability to support the definition of the dynamic behavior of a system (along the lines of activity graphs).

4.2 Identification and Resolution of Conflicts and Inconsistencies

Discussion members: *Lodewijk Bergmans (University of Twente, The Netherlands), Pascal Durr (University of Twente, The Netherlands), Pavel Hruby (Microsoft Business Solutions, Denmark), Andrzej Krzywda (Wroclaw University, Poland), Awais Rashid (Lancaster University, UK), Pawel Slowikowski (AGH University of Science and Technology, Poland)*

Motivation: AOSD allows for new ways to structure software. While the new constructs offer different ways to implement concerns (especially crosscutting ones), aspect definitions and base concerns can conflict with each other or themselves. There is a need to identify and categorize the potential inconsistencies and conflicts that can occur in aspect-oriented software, in order to provide a list of issues that developers should be aware of. Resolution of the inconsistencies also needs to be investigated.

Categories of Conflicts: The following four categories of conflicts and inconsistencies in AO software were identified:

1. **Crosscutting specifications:** Currently, aspect-oriented approaches specify crosscutting in terms of join points in the base concerns. This can lead to two problems: *accidental join points* and *accidental recursion*. The former implies accidentally matching unwanted join points (e.g., when using wild cards in AspectJ) and thus applying the aspect's behavior at the wrong places. As discussed in section 3, intentional join points can make this problem easier to detect and avoid. Accidental recursion refers to the situation when the aspect behavior itself matches a join point specification description leading to recursion: a join point is matched, its associated aspect behavior is applied, which in turn matches the join point again, and so forth.
2. **Aspect-aspect conflicts:** When multiple aspects exist in a system, they can conflict with each other (which is also called *aspect interaction*). We can identify five categories of interactions: *conditional execution* where applicability of one aspect is dependent on another aspect being applied; *mutual exclusion* when composing one aspect implies that another one must not be composed; *ordering* required when aspects influence the same point in the base concerns; *dynamic context dependent ordering* which differs from simple ordering in that aspect ordering depends on the dynamic state of the system

or the context in which the aspects are being applied; *tradeoffs and conflicts at requirements and architectural level* where aspects influencing the same elements can result in having to compromise particular requirements in favor of others.

3. **Base-aspect conflicts:** Aspects can conflict not only with each other but also with the base concerns. These kinds of conflicts arise when the base concerns explicitly refer to or depend on aspect behavior (i.e., when aspects are invasive). This can lead to *circular dependencies between aspect and base* and the need for *base to communicate with the aspect.*
4. **Concern-concern conflicts:** Similar to the previous categories, conflicts can also occur between concerns. These kinds of problems arise when concerns affect the execution or state of other concerns. The *change of functionality* made possible by aspects can impact other modules. For instance, at some point in program execution one aspect would normally be applicable, but because of another aspect the proper join point is never reached. *Inconsistent behavior* can occur when an aspect destroys or manipulates the state of another aspect or the base concern. *Composition anomalies* can arise e.g., problems with subtype substitutability.

General Issues and Open Questions: The identification and resolution of conflicts and inconsistencies in AO software requires extensive investigation. When undertaking such investigation, the following issues need to be explored:

1. *Should aspects be allowed to break encapsulation?* Many AO programming approaches offer the possibility to break encapsulation of base code units, such as classes. While this generally gives the developer more flexibility and power, it also produces aspects that are more brittle, in that they rely on internal implementation details and not just on the available interfaces. Changing base code would then require taking into account the aspect definitions in the system, which would contradict the idea that aspects should be non-invasive
2. *Is there a 'fragile base aspect' problem?* The question here is whether use of inheritance to reuse a base aspect's behavior leads to anomalies similar to the fragile base class problem in OO systems where seemingly innocuous modifications to the base class code can have an undesirable effect on its sub-classes. Are there better means to promote reusability and variability of aspects than the use of inheritance?
3. *How should one deal with incremental changes?* Incremental changes introduce two problems. Firstly, they can leave the system temporarily in an inconsistent state, breaking the interaction of base- and aspect behavior with each other or themselves. Secondly, they can unintentionally cause other parts of the system to break. Impact analysis of changes in AO systems is discussed in section 4.3.

4.3 Impact Analysis for Aspect-Oriented Software

Discussion members: *Adrian Coyler (IBM Hursley Labs, UK), Ouafa Hachani (Equipe SIGMA, France), Jan Hannemann (University of British Columbia, Canada), Stein Krogdahl (University of Oslo, Norway), Maximilian Stoerzer (Universität Passau, Germany)*

Motivation: Software systems change. If a change is to be made to a system, it is necessary to know what other parts of the system are affected by it, to either automatically adjust those parts, or to point the developer to potential problems. Refactoring is a disciplined way to change the structure of software systems while preserving their behavior. While refactoring is well understood for object-oriented software, AO constructs represent additional ways to structure software and as such require additional considerations when determining the impact of a change. There is a need to explore the specific problems that can occur when an AO system is changed. Such an impact analysis is useful for semi-automated refactorings. It can potentially help a user to choose between implementation alternatives, and can determine the effects of weaving an aspect. Note that we only focus on static analysis here, since dynamic analysis is often much harder to employ.

Three main research questions need to be addressed:

1. What are the potential consequences of a change and the problems caused by it?
2. Which consequences are detectable (first step) and which can be analyzed to be safe (second step)?
3. What can be done to make impact analysis easier?

Impact Analysis: In general, we have different cases: modifying (non-aspect) code that is not advised, modifying code that is, and modifying aspect code. The first case seems to be the least problematic and falls into the realm of OO impact analysis, while the other two are discussed here. Table 1 presents a categorization of the consequences (or problems) and their potential effect on the system. For each problem category, we indicate whether that particular problem can be detected and analyzed. Detection refers to the ability of a static analysis tool to identify the potential problem, while an analysis would allow deciding whether the consequences are "safe", i.e. do not adversely affect other parts of the software. As the table shows, it is often possible to detect whether a particular change has the potential to adversely affect the system. Unfortunately, in many cases it is impossible to analyze whether such a change is safe or not, as that usually requires analyzing the dynamic impact of arbitrary code blocks. In these cases, it is more feasible to have human intervention make the decision, i.e., asking the developer to appraise the situation.

Special Remarks: It turns out that for aspect interference, it is not sufficient to say that a problem only occurs if we have the same JPs AND the same data is manipulated. For example, *billing* and *timing* (telecom example) have the same JPs, but do not manipulate the same data. The dynamic method lookup and the dynamic behaviour of `instanceof` constructs can be changed by `declare parents` statements. For example, assume we have a class hierarchy $A < B < C$. If C is declared to be a subclass of A, then C's dynamic method lookup will omit B.

Table 1. Categorization of the consequences and their potential effect on the system

Problem / Consequence	Causes	Detectable?	Analysis possible
Change in advice matching, both static and dynamic.	Renaming, moving, splitting, or joining of methods affects PCs that rely on them	Static: yes Dynamic: no	Cannot analyze im–pact of arbitrary code blocks. Human intervention required
Aspect interference	Multiple matches at one joinpoint can cause unwanted behavior if the ordering is not specified	yes	If precedence is de–clared, analysis is easy. Otherwise hu–man intervention is required
Declare precedence	Using this construct influences all places where JPs are shared. Not possible to have different ordering at different JPs	yes	Trivial case: no mul–tiple matches. Other–present user with multi–matches
Around advice	Can change arguments, control flow, return value	Fuzzy boolean —it is easy to say never or maybe	Very hard – human intervention
Declare parents	Can change dynamic method lookup and dy–dynamic behavior of instanceof constructs	yes	Trivial cases: no overridden members, no instanceof tests
Inter-type declarations (introductions)	Can hide members and change dynamic method lookup	yes	yes
PC used in perXXX aspect (pertarget, percflow, ...)	Changing the PC may impact the number of aspects created	yes	no
Declare soft	Can create problems for future changes if exceptions are not caught	yes	yes

Making Analysis Easier: To make impact analysis easier, we have the following suggestions, based on our findings above:

1. Provide different types of aspects: add invariants that can be statically analyzed. This may reduce the scope of checking. For example, an aspect that does not affect other parts of the system is much less problematic than one that does (potential examples: logging, timing, billing). All such changes should be part of the language.
2. Involve the user in decisions: tools may be able to "learn".

From a programmer's perspective, it is also possible to use assertions and proper documentation and check them by hand to help ensure that new/changed aspects will not break anything (or to identify places where they do). This however puts the burden on the developer and can not be checked by a static analysis tool.

Other Issues: Similar difficulties of analysis can be found in parallel programming (different concerns, interwoven control flow). Note that tool support can help if no impact analysis tools are available. For example, if base code gets replaced with a PC and advice, ajdt allows (or will allow) comparing original code and the PC matches. There are other (non-technical) impacts that one should be aware of, such as introducing a new aspect that breaks the (intended) encapsulation of another class. These consequences are generally not detectable or analyzable by a tool without additional language support.

5 Summary and Future Research

The discussion in this report has highlighted a number of promising approaches to analysis of AO software. At the same time, it has made clear the need for extensive further research in this area, as this will be crucial to the widespread acceptance of AO concepts, tools and techniques.

One of the key points raised during the panel and group discussions is the need to intentional means to specify the join points to which an aspect's behavior may apply. We believe this will be beneficial both in understanding aspect-oriented software and in increasing the robustness of aspect behavior with respect to base-code modifications.

Other issues are the identification and resolution of conflicts and inconsistencies and analyzing the impact of changes in an AO system. Effective support (both models and tools) for this purpose is crucial to the maturity of AO techniques. Last, but not least, non-invasiveness is an important property for aspect, which, if preserved, can significantly aid analysis. There is a need to focus on AO approaches which are non-invasive, have more intentional means for specifying base-aspect relationships and can handle conflicts and inconsistencies within a coherent, well-defined framework.

The present discussion has also taken the first steps in general categorizing of conflicts and inconsistencies in AO software, and more closely considering the consequences and effects of change to AO software for a specific language environment (AspectJ).

References

1. Lodewijk M. J. Bergmans. Towards detection of semantic conflicts between crosscutting concerns, presented at: ECOOP Workshop on Analysis of Aspect-Oriented Software, 2003.
2. Andy Clement, Adrian Colyer, and Mik Kersten. Aspect-Oriented Programming with AJDT, presented at: ECOOP Workshop on Analysis of Aspect-Oriented Software, 2003.
3. Constantinos A. Constantinides. A Case Study on Making the Transition from Functional to Fine-Grained Decomposition, presented at: ECOOP Workshop on Analysis of Aspect-Oriented Software, 2003.
4. Sivia de Castro Bertagnolli and Maria Lucia Blanck Lisboa. The FRIDA Model, presented at: ECOOP Workshop on Analysis of Aspect-Oriented Software, 2003.
5. Ouafa Hachani and Daniel Bardou. On Aspect-Oriented Technology and Object-Oriented Design Patterns, presented at: ECOOP Workshop on Analysis of Aspect-Oriented Software, 2003.
6. Jan Hannemann, Ruzanna Chitchyan, and Awais Rashid. ECOOP Workshop on Analysis of Aspect-Oriented Software, `http://www.comp.lancs.ac.uk/ computing/users/chitchya/AAOS2003/ AAOS_Home.php`, 2003.
7. A. M. Reina, J. Torres, and M. Toro. Aspect-Oriented Web Development vs. non Aspect-Oriented Web Development, presented at: ECOOP Workshop on Analysis of Aspect-Oriented Software, 2003.
8. Pawel Slowikowski and Krysztof Zielinski. Comparison Study of Aspect-Oriented and Container Managed Security, presented at: ECOOP Workshop on Analysis of Aspect-Oriented Software, 2003.
9. Maximilian Stoerzer, Jens Krinke, and Silvia Breu. Trace Analysis for Aspect Application, presented at: ECOOP Workshop on Analysis of Aspect-Oriented Software, 2003.

Modeling Variability for Object-Oriented Product Lines

Matthias Riebisch, Detlef Streitferdt, and Ilian Pashov

Technical University Ilmenau,
Max-Planck-Ring 14, 98684 Ilmenau, Germany
{matthias.riebisch|detlef.streitferdt|ilian.pashov}@tu-ilmenau.de

Abstract. The concept of a software product line is a promising approach for increasing planned reusability in industry. For planning future requirements, the integration of domain analysis activities with software development for reusability turned out to be necessary, both from a process and from an economic point of view. In this context, variability of requirements in a domain is expressed by feature models. Feature models enable planning and strategic decisions both for architectural and for component development. By expressing feature dependencies, feature models are used to partition the architecture and the implementation. For industrial use, appropriate methods for modeling variability in requirements, design and implementation as well as tools for supporting feature models and for integrating them with other models are needed. The ECOOP workshop explored the possibilities and limitations of feature models and supporting methods. Its fully reviewed contributions aim at improving the feature model usage as well as the integration into the software development process. Improving industrial applicability of feature modeling and methods is an important goal. This paper provides a summary of the discussion and presents the major results as well as important questions and issues identified for future research.

1 Introduction

The approach of developing software for industrial reusability was one of the driving forces of the concept of software components and of application frameworks. The software product line approach represents a successor and an extension of these concepts by planning reusability based on common requirements. For this purpose, domain analysis concepts are introduced as a complementary technology to requirements analysis. By analyzing the domain of a whole family of software products, the decisions for developing a common architecture of these products and common components can be made properly. For these decisions information about commonality and variability of the requirements of the domain is needed.

Domain Analysis methods provide this type of information. The method Feature Oriented Domain Analysis FODA [10] introduced the feature model as a means of expressing commonality and variability of requirements as well as dependencies between them.

Since then, a variety of successors have been developed as described in the next section of this paper. They are all based on the idea of features as user visible

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 165–178, 2004.

properties of products. In the product line approach, features are applied for the distinction of products of a product line, as well as for configuring them. In a feature model features are structured in a hierarchy.

Feature models offer some advantages. They enable easy navigation and comprehension of requirements - especially as an addition to requirements in natural language descriptions. Their abstraction level is closer to the system architecture than the one of requirements. Therefore features bridge the abstraction gap between requirements and system architecture. This fact improves the understanding and supports the communication between the different participating roles. Furthermore, features are structuring the requirements thus enabling checks for completeness and for achievement as well as easier effort estimation, compared to requirements as a structured text.

The workshop was held

- to strongly integrate feature modeling as a part of the software engineering process
- to improve the feature model definition (syntax and semantics) in order to leverage the development activities
- to expand industrial applicability of feature models
- to introduce tool support.

The workshop discussions and the full papers [2], [14], [15], [17], [19], [22] and [23] as published in the workshop proceedings [21] have been a contribution to this paper.

2 State of the Art

The established means of describing software systems such as models for data structures, architectures and behavior aim at developing concrete systems without variability. For pre-fabricating reusable software, especially by software product lines, variability is needed for deciding about common properties and parts.

Domain analysis investigates commonalities and variabilities of products within a domain. The Feature Oriented Domain Analysis method FODA [10] describes common and variable features in a hierarchy of properties of domain concepts. The properties are relevant to end users. At the root of the hierarchy there is a so-called concept feature, representing a whole class of solutions. Underneath the concept feature there are hierarchically structured sub-features showing refined properties. Each of the features is common to all instances unless marked as being optional, thus not necessarily being part of all instances. Fig. 1 shows an example for an ATM product line with a feature `ATM` as concept feature. The feature `debit card reader` is a so-called mandatory feature, stating that it is common to all instances of the domain, because every ATM has a reader for debit cards. The feature `receipt printer` is marked as optional by an empty bullet, because there are ATMs in this product line example without a printing device.

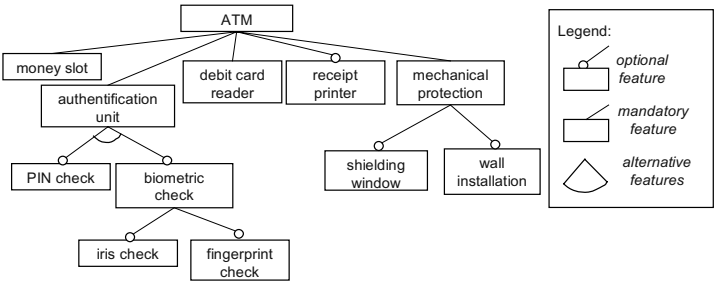

Fig. 1. Feature Model Example

Such a feature model represents an abstract view onto properties of all instances of a domain. Every feature covers a set of requirements. By selecting a set of optional features an instance of that domain can be defined. By definition all mandatory features are part of the instance. Dependencies between features are described by relations with attached explanatory texts.

The successor methods FORM [11] and its successor FOPLE [12] introduced four different views with features classified to the according types:

1. Capability features:
   Service, Operation, Non-functional characteristics.
2. Domain technology features:
   Domain method, Standard, Law.
3. Operating environment features:
   Hardware, Software.
4. Implementation technique features:
   Design decision, Communication, ADT.

These views are intended for improved comprehension and better navigation. Unfortunately there is no clear definition for making a distinction between these views. User-visible properties should be covered by the Capabilities view, however some of the properties of this category like graphical user interface components could even be assigned to the Operating Environment view. A service could be designed according to an existing act and therefore assigned to the Domain Technology or to the Capability view. Due to the missing definition of the views, ambiguities can occur, and possibilities for tool support are limited. Thus, the advantages of these views do not result in support for managing larger sets of features.

For planning software reusability, several approaches have been developed for inclusion of feature models into software development methods. The method FeatuRSEB connects FODA with the Object-oriented approach of the method Reuse Driven Software Engineering Business RSEB [9]. Features are applied to control variability of the software architecture. UML is extended by variation points to model software product lines. Use cases express the requirements. However, the method describes a large portion of the necessary information by informal means, therefore possibilities for tool support are limited. The feature models in this approach are extended by two types of alternatives OR and XOR. The graphical notation is lightly changed compared to FODA. Other significant extensions compared to FODA are

cardinalities and associations between features. [23] contains an comparison of the different graphical notations.

Czarnecki and Eisenecker extend the alternatives of the types OR and XOR by the "n of many" selection and change the graphical notation slightly, compared to FODA [6]. Premature features are introduced to express vagueness.

Bosch extends the feature model by an so-called external feature to improve the description of variability of a software architecture [3]. Furthermore, he introduces a different graphical notation for OR and XOR alternatives.

For preventing ambiguities and to enable a more expressive and powerful notation for relations between neighboring features, multiplicities have been introduced by our own works [20]. These multiplicities are similar to those of Entity Relationship Models ERM and of the Unified Modeling Language UML. In addition to joining some features to a group, all of them are designated as optional to express the possibility of choice.

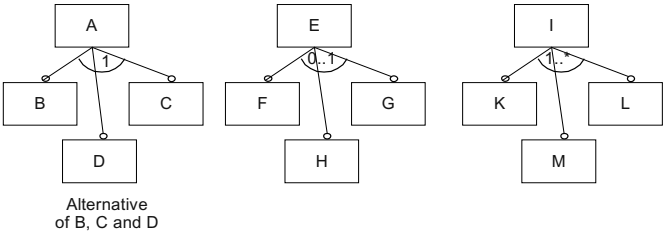

Fig. 2. Grouping neighboring features using multiplicities

Different variants of multiplicities of the groups are possible:

- 0..1 at most one feature has to be chosen from the set of the sub-features,
- 1 exactly one feature has to be chosen from the set,
- 0..* an arbitrary number of features (or none at all) have to be chosen from the set,
- 1..* at least one feature has to be chosen from the set,

Situations in which a set of features has a multiplicity like "0..3", "1..3", or simply "3" are possible as well.

Summarizing we have to encounter that there is no clear definition of the description of feature models, their categories and their relations. Different notations, informal expressions of relations and dependencies as well as ambiguities limit the applicability of methods and tools. However, for industrial use of feature models, improved support by methods and tools is crucial for success. Therefore a more strict *and* comprehensive definition of feature modeling is needed. It should serve as the base for an improved methodical support for the software engineering process, especially for variability management and product derivation based on feature models. Currently, there is no single method allowing to use the full potential of feature modeling.

3 Issues of Discussion

The workshop participants identified two clusters of open questions to improve the situation described above. The first cluster covers questions of definition and usage of feature models, views covered, possible extensions and their relationship to requirements and implementation. The issue of complexity and of how to manage is of great concern. These questions are accompanied by issues of tool support. The role of parameters and of non-functional properties are covered as well. The issues were reduced to more comprehensive questions, driven by the subjects of interest of the participants:

- Features across different families.
- Feature views and categories
  - 4 views of FORM [11] – "… are informal, to support developers with feature modeling tasks".
  - Customer view – for the selection of features for a desired application.
  - "Triangle" of Needs : Features : Requirements for top-down analysis
- How complex should feature models be, how detailed ?
  - Complexity depends on „wisdom" of the developer, on the number of features, number of dependencies, size of the system to be modeled
  - Complexity should be reduced by the feature model.
  - Complexity depends on the offered variability.
- Tool implementation of feature views
- Usefulness of extra functionality (special term for non-functional properties)
  - Everything not representable by features is an extra functionality
  - What about parameter features?
- Ways of acquiring feature models

The second cluster covers questions of handling feature models, of decision making based on them and of identifying variability by separating common and variable elements. Methodical ways of designing solutions based on feature models - both in forward and reverse engineering - were discussed:

- What design patterns are relevant for implementing variability
- How to recover design decisions in existing implementations based on feature models
- Mapping between feature models and design models
- Relations, cardinalities of requirements with feature models and design models
- Separation of common and variable parts
- Object-oriented aspects of modeling variability
- How to handle refactoring
- Model formalization

4 Approach

4.1 Role of Feature Models

In our opinion, feature models should bridge the gap between requirements and the solution. They provide an extra model between requirements specifications (i.e. structured text with glossaries, concept graphs, use case models, decision models etc.) and design models and architectures (UML models, ERM models). In our opinion it is not desirable to extend feature models by more information, i.e. with the goal to replace some parts of these well-established descriptions for requirements and design.

According to our experiences from industrial application of feature models, they can successfully support some product line development activities of two groups of stakeholders. The first group, handling software more as a black box, consists of customers, merchants, product managers and company managers. For them, a feature model

- provides an overview over requirements
- distinguishes between common and variable properties
- shows dependencies between features
- enables feature selection for defining new products
- supports decisions about evolution of a product line

The second group of stakeholders is working on the development of a product line, i.e. architects, software developers for reusable components of the product line as well as developers for single products. They are supported by feature models in

- defining reusable components and separating them according to the Separation of Concerns principle
- assigning reusable components to variable features
- describing dependencies and constraints between components and features
- controlling the configuration of products out of the reusable components

By linking features to elements of design and implementation, additional information about details of the solution domain are provided. By linking features to requirements, detailed information from the problem domain is reachable. These links are built using traceability links [24]. Fig. 4 shows this linkage by an ERM diagram.

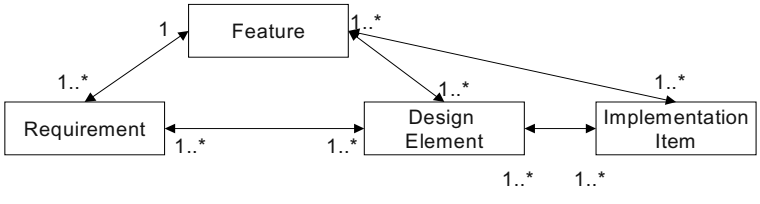

Fig. 3. References between features, requirements, design and implementation

Based on these goals and on the linkage to other models, a customer view is sufficient for the feature model. All other information should be captured in the established descriptions, i.e. requirements specification, design models and implementation documents. A feature model provides an overview of the requirements, and it models

the variability of a product line. It is used for the derivation of the costumer's desired product and provides a hierarchical structure of features according to the decisions associated with them.

Separation of concerns represents an important software engineering principle. Current design and implementation methodologies can only partly reach this principle. Feature modeling supports the goal by structuring the relations between requirements and system architecture. A situation as shown in fig 4 – with *only one* architectural element per feature and per implementation element - would be ideal for supporting evolution by tracking the impact of a change, as well as for configuring new products by recombining features. Furthermore, such a simplification of the relations would lead to easier navigation and comprehension of a software system. Compared to fig 3, redundant relations could be omitted without a loss of information. However, this situation is not reachable in practice because of limitations of several reasons, i.e. the implementation technology, the human factor, the insufficient tool support and the complexity of the software systems and their environment.

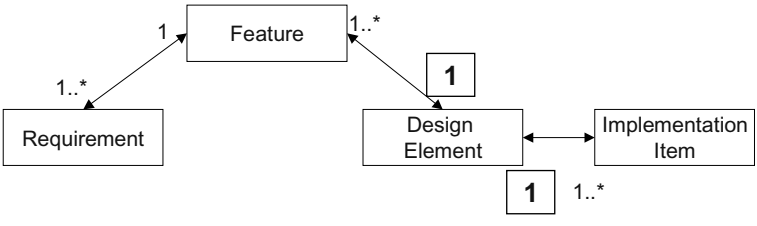

Fig. 4. Idealistic relations between features and design with a complete separation of the feature concerns

4.2 Feature Categories and Definition

From this point of view, we agree on the definition of [10] where, a feature is "a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems" and of [8], where "a feature model is a specific domain model covering the features and their relationships within a domain".

A feature represents a “property of a domain concept … used to discriminate between concept instances”. For the case of a software product line, concept instances are the products of that product line. Features can be of different types, where non-functional features play a special role for a product line architecture [2].

According to this definition, not only features describing capabilities of a system are possible. Even very technical concepts - i.e. “common rail fuel injection” for a car engine - can occur as features, if there is the chance that customers will use these concepts for distinguishing between products. In our experience, the decision about including a concept as a feature becomes very clear by asking if a customer is willing to pay for it.

Definition – Part 1
A feature represents an aspect valuable to the customer. It is represented by a single term. There are three categories of features:
- *Functional features* express the behavior or the way users may interact with a product.
- *Interface features* express the product's conformance to a standard or a subsystem
- *Parameter features* express enumerable, listable environmental or non-functional properties.

A feature model gives a hierarchical structure to the features. In addition to the mentioned categories, within the hierarchy there could be abstract features to encapsulate *Concept features*. The root of the hierarchy always represents a concept Feature.

The four categories were defined aiming at a small number of categories and at an unambiguous distinction between them. *Functional features* describe both static and dynamic aspects of functionality. They cover i.e. use cases, scenarios and structure. To give some examples for the automotive domain, features like `Electric seat heating` and `Extra daytrip mileage counter` belong to that category.

Interface features describe connectivity and conformance aspects as well as contained components. From the customers point of view the possibility of connecting a product to other devices and of extending it are valuable categories. Examples for features from this category are `Firewire connection` for an electronic camera and `DDR133 RAM` for memory sockets of a PC. Conformity to standards and certificates are in this category as well, i.e. `USB 2.0 compatible` and `ISO 9000 certified` for a PC. Complete components or subsystems of special quality or by special vendors were added to the same category, because the handling of such features is very similar to interfaces. An example is the feature `Bosch ABS device` for a car, if this is valuable for a customer.

Parameter features cover all features with properties demanding for quantification by values or for assignment to qualities. Examples from the automotive domain are `fuel consumption`, `top acceleration` or `wheel size`.

Concept features represent an additional category for structuring a feature model. Features of this category have no concrete implementation, but their sub-features provide one. The feature `mechanical protection` in fig. 1 represents an example for such a feature.

4.3 Relations in a Feature Model

Within a feature model the features are structured by relations. Common to all methods mentioned above are hierarchical relations between a feature and its sub-features. This kind of relation controls the inclusion of features to instances. If an optional feature is selected for an instance, then all mandatory sub-features have to be included as well, and optional sub-features can be included. Additionally, FODA introduces so-called composition rules using "`requires`" and "`mutex-with`". These rules control the selection of variable features in addition to the hierarchical

relations. If a feature A is selected for an instance, and there is a relation "`A requires B`" then feature B has to be selected as well. Opposite to this, if a feature A is selected for an instance, and there is a relation "`A mutex-with B`" then feature B has to be unselected. In Generative Programming the latter relation is called "`excludes`" instead of "`mutex-with`".

When applying these relations in practical projects, some deficiencies become visible. When building a hierarchy of features it is not clear how to arrange the features. Frequently it was not obvious whether to express a particular relation between two features by assigning one as a sub-feature of the other or by establishing a "`requires`" relation between them. Semantically, there are only small differences between a feature - sub-feature relation in a hierarchy and a "`requires`" relation. The same was observed with alternatives and "`mutex-with`" (see fig. 5). There is only little support for decisions between these possibilities for a relation. The description of more complex dependencies, with more than two participating features or more conditions is impossible.

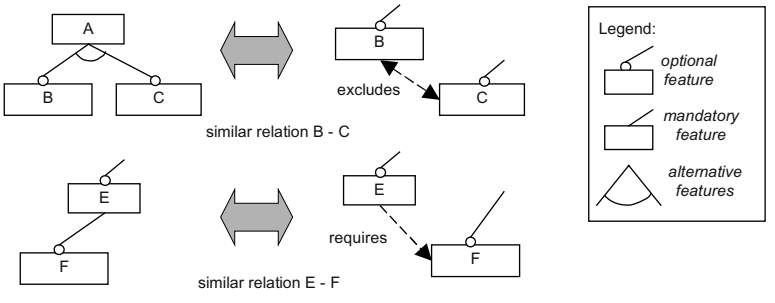

Fig. 5. Similar semantics of hierarchical relations and dependency relations

Feature models are established during Domain Analysis. Similar to the requirements analysis, this step usually consists of the activities elicitation, modeling and verification [13]. To meet the business goals of time and budget, scoping decisions during each of these activities have to be performed. While establishing a domain model and a feature model, dependencies between features have to be detected. If mutual interactions of behavior and properties of features can be dissolved, the variability of a product line can be enhanced and the configuration of its products can be simplified [4].

For existing systems or components, feature models can be established by reverse engineering the system's architecture and requirements. Program comprehension and document analysis serve as the base for feature model-based approaches [18] which recover the influence of features to the system architecture and enable refactoring and architectural evolution.

Definition – Part 2
The features are structured by relations of the following categories:

- *Feature - sub-feature - relations* construct a hierarchy, designed to guide the customer's selection process. The position of a feature within the hierarchy shows its influence on the product line architecture. A hierarchy relation could carry the semantics either of the *requires* relationship or of the *refinement* one. The hierarchy relations distinguish between *mandatory* and *optional* features.
- Constraints between features are expressed either by *multiplicity-grouping relations* for features with the same parent or by *requires* or *excludes* relations for arbitrary features.
- *Refinement relations* lead towards more detailed sub features. They express is-a or part-of semantics.
- Suggestions for additional selections of features dependent on other features can be expressed by the *recommends relation*. This relation represents a weak form of the *requires* relationship. It can also be used for expressing a default value for a feature's selection.
- As a weak form of exclusion, the *discourages* relation represents the counterpart of the latter.

Requires, excludes, refinement, recommends and *discourages* relations bridge arbitrary features within the hierarchy.

4.4 Application of Feature Models

Feature models contribute to efficiency and structure of various software development activities. Three of the activities of the two groups of stakeholders mentioned in the beginning are examined a little closer: the definition of a new product out of the product line by a product manager, the configuration of this product by a developer and the evolution of the product line's platform during its extension.

A new product of a product line is defined by selecting a set of optional features from the feature model. This could be done by a product manager, a customer or by another stakeholder related to them. Following a top-down procedure, a decision for each variable feature is made whether to include it or not. All mandatory features are part of the product by default. All sub-features of a not selected variable feature are not included into the new product even if they are mandatory features. If no decision is made, the default value given by the *recommends* relation is applied. While selecting an optional feature, its relations to other features i.e. *requires* or *excludes* have to be considered. This activity results in a set of features this particular new product will offer.

The selection of features is strongly related to the configuration of the structure of the resulting product. This is done by a software developer, supported by tools. If an optional feature is selected, the design and implementation components related to this feature (see fig 2) have to be part of the design and implementation of the particular product. The value assigned to a parameter feature is passed to the corresponding parameter of the implementation. If all dependencies and constraints between components are expressed as relations between features, this activity will result only in valid product configurations. To avoid errors, the application of tool support during this activity is crucial.

In many cases, new products demand for new features or new combinations of available ones. Thus, the pre-fabricated reusable platform of this product line has to be evolved. Based on the feature model, needed extensions and changes are detected. Business decisions about extensions and changes are supported by the information about their impact on existing assets, provided by the feature model and its traceability links to design and implementation (see fig 2). In the case of an addition of properties, components are added or extended. In the opposite case of discarded properties, refactoring of the reusable platform or of single components are necessary. In some cases, constraints between components have to be considered to enable new

combinations of variable features required by the customer. After removing these constraints, the feature model has to be changed accordingly.

5 Tool Support

As a result of the workshop discussion the group agreed on the point, that user acceptance of feature modeling is dependent on the available tool support. The number of features, the relations between the features and other design elements might be too complex for manual processing. Thus, tool support was identified as an important issue of feature modeling. On the one hand tools for feature diagrams are needed to make use of the improved graphical overview feature diagrams offer. Graphical feature editors such as AmiEddi [1] are capable of modeling simple feature diagrams but need further development to support all the aspects of the aforementioned feature definition. Cardinalities can only be used in a basic form and complex interrelations between features can neither be modeled nor checked with the tool. Other approaches for the graphical representation make use of the UML as in [7], where feature models are expressed by UML class diagrams, with stereotypes changing classes into features. In our opinion this approach is good for a prototypical implementation, but is lacking the clearness and expressiveness of real feature diagrams. Using UML class diagrams with stereotypes totally changing their intention and meaning causes additional misunderstandings during the development of product lines.

There is a UML-based prototype tool xFIT exploiting feature models for semi-automatically generating product line instances by framework instantiation [16]. By using XMI, the diagrams can be integrated to commercial CASE tools.

However, the above mentioned tools only support very basic consistency checks for feature models. In most of them, only the hierarchy of a feature model of the product line can be checked, based on the construction process of the tools. The tools are able to model the requires and excludes relations. Such relations need to be checked for ever variant that is derived out of the product line and the tools are not able to recognize a broken relation in a way, that developers can refine the model. Furthermore the tools don't provide mechanisms to introduce new relations, as described in our feature definition. Thus, further development effort need to be put into these or new tools to fully support the use of feature diagrams and into feature model checkers, with the capability to validate variants of the product line. A very important issue of feature modeling are the traces of features to other design elements. In this field no tool support is available. By the usage of requirements engineering tools like Requisite Pro or DOORS traces can be erected and handled in trace tables, but features are not handled by these tools and no other tools are currently available, that can process traces between feature models and other design elements. However, open tools i.e. AmiEddi's successor CaptainFeature [5] offer a good potential for such extensions. Given the XMI data format of the UML and the support of an XML export of AmiEddi or CaptainFeature further research efforts are needed to develop tools for the processing of XML trace links between feature models and UML design elements.

Finally all tool development effort should lead towards the integration of feature modeling into current CASE tools. It is vitally important to enable developers to

further use their "old" development tools and at the same time enhance the development of product lines with feature models. In this respect the currently available tools are just at the beginning of the development of add-ins for CASE tools for professional usage.

6 Experiences

Addressing industrial applicability is one of the major concerns of the workshop. Therefore, experiences with feature modeling in research and industry were discussed. The benefits of the use of feature models for the time-to-market have been examined, compared to other ways of developing software systems. One example for practical use is a product line of car periphery systems with about 200 features. The features were modeled using the tool DOORS and then mapped into a configuration tool. The modeling has taken 1 month, and it lead to an effective configuration support for the product line.
As another example a yard inventory system for steel manufacturing was discussed. For expressing its variability, 1.500 features were introduced. For the modeling one year was necessary, however non-functional features were not explicitly covered. It was estimated that the most important benefit of feature modeling results rather from the improved time-to-market by configuration support than from a cost reduction.

7 Questions for Future Research

As a result of this workshop, questions for future research have been identified:

- Traceability links play an important role both for connecting features to implementation and for identifying the impact of change and configurations.
- According to the usage of feature models in defining new product line instances, a change of the structure of the hierarchy is necessary, if the importance of single features has changed.
- Expressing non-functional features and their connection to other features needs more clarification.
- The maintenance of feature models themselves is very important for keeping the consistency of a product line. Therefore refactoring methods and management of typical structures in feature models are to be investigated.
- Strategies and a process model for product line evolution has to be established to describe the central role of feature models.
- The current works on model-driven development have to be extended by feature modeling.
- Tool support represents a factor critical for the success of a product line's evolution. Furthermore, the introduction of such a process into an enterprise needs further consideration.

Acknowledgements. This paper reflects results of our own research as well as the results of the ECOOP'2003 WS „Modeling Variability for Object-Oriented Product Lines". All Participants: David Benavides, Paulo Borba, Ron Crocker, Kai Hemme-Unger, Lothar Hotz, Thorsten Krebs, Jaejoon Lee, Toacy C. Oliveira, Ilka Philippow and Silva Robak as well as the authors of this paper contributed to the workshop by problem statements, proposals, experience reports and questions. The paper presentations of this workshop were published in [21], they are available in the bookstores. We want to thank all participants for their work. We want to thank the program committee members for their reviews of the presented papers.

References

1. Lang, M.: AMIEDDI, http://www.informatik.fh-kl.de/~eisenecker/projekte.html.
2. Benavides, D.; Ruiz-Cortés, A.; Corchuelo, R.; Durán, A.: Seeking for Extra-Functional Variability. In: [21] pp. 58-63.
3. Bosch, J.: Design and use of software architectures - Adopting and evolving a product-line approach. Addison Wesley, 2000.
4. Calder, M.; Kolberg, M.; Magill, M.H.; Reiff-Marganiec, S.: Feature Interaction - A Critical Review and Considered Forecast. Elsevier: Computer Networks, Volume 41/1, 2003. S. 115-141
5. CaptainFeature. Opensource Feature Modeling Tool, Available online at https://sourceforge.net/projects/captainfeature/
6. Czarnecki, K., Eisenecker, U.W.: Generative Programming. Addison Wesley, 2000.
7. Clauß, M.: A proposal for uniform abstract modeling of feature interactions in UML, In Proceedings of the European Conference on Object-Oriented Programming (ECOOP), 2001
8. Ferber, A.; Haag, J.; Savolainen, J.: Feature Interaction and Dependencies - Modeling Features for Re-engineering a Legacy Product Line. in Proc. 2nd Software Product Line Conference (SPLC-2), San Diego, CA, USA, (August 19-23 2002). Springer, Lecture Notes in Computer Science, 2002, pp. 235- 256.
9. Jacobson, I; Griss, M.; Jonsson, P.: Software Reuse Architecture, Process and Organization for Business Success, acm Press, 1997
10. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A., Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1990.
11. Kang, K., Kim, S., Lee, J., Kim, K., Shin E. and Huh, M.: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures, Annals of Software Engineering, 5, 1998, pp. 143-168.
12. Kang, K.C.; Lee, K.; Lee, J.: FOPLE - Feature Oriented Product Line Software Engineering: Principles and Guidelines. Pohang University of Science and Technology, 2002
13. Kotonya, G.; Sommerville, I.: Requirements Engineering - Processes and Techniques, 1998.
14. Krebs, T.; Hotz, L.: Needed Expressiveness for Representing Features and Customer Requirements. In: [21] pp. 23-31.
15. Lee, J.; Kang, K.C.: Feature Binding Issues in Variability Analysis for Product Line Engineering. In: [21] pp. 77-82.
16. Oliveira, T.C., Alencar, P., Cowan, D., Towards a declarative approach to framework instantiation Proceedings of the 1st Workshop on Declarative Meta-Programming (DMP-2002), September 2002,Edinburgh, Scotland, p 5-9

17. de Oliveira, T.C.; Filho, I.M.; Alencar, P.; de Lucena, C.J.P.; Cowan, D.C.: Feature Driven Framework Instantiation. In: [21] pp. 1-22.
18. Pashov, I., Riebisch, M.: Using Feature Modeling for Program Comprehension and Software Architecture Recovery. In: Proceedings 10th IEEE Symposium and Workshops on Engineering of Computer-Based Systems (ECBS'03), Huntsville Alabama, USA, April 7-11, 2003. IEEE Computer Society, 2003, pp. 297-304
19. Philippow, I.; Streitferdt, D.; Riebisch, M.: Design Pattern Recovery in Architectures for Supporting Product Line Development and Application. In: [21] pp. 42-57.
20. Riebisch, M.; Böllert, K.; Streitferdt, D., Philippow, I.: Extending Feature Diagrams with UML Multiplicities. 6th World Conference on Integrated Design & Process Technology (IDPT2002), Pasadena, CA, USA; June 23 - 27, 2002.
21. Riebisch, M.; Coplien, J.O.; Streitferdt, D. (eds.): Modelling Variability for Object-Oriented Product Lines. BookOnDemand Publ. Co., Norderstedt, 2003. ISBN 3-8330-0779-6.
22. Riebisch, M.: Towards a More Precise Definition of Feature Models. In: [21] pp. 64-76.
23. Robak, S.: Modeling Variability for Software Product Families. In: [21] pp. 32-41.
24. Sametinger, J.; Riebisch, M.: Evolution Support by Homogeneously Documenting Patterns, Aspects and Traces. 6th European Conference on Software Maintenance and Reengineering. Budapest, Hungary, March 11-13, 2002 (CSMR 2002) . Computer Society Press, 2002. S. 134-140.

Object Orientation and Web Services

Anthony Finkelstein[1], Winfried Lamerdorf[2], Frank Leyman[3], Giacomo Piccinelli[1], and Sanjiva Weerawarana[4]

[1] Department of Computer Science, University College London, United Kingdom
{A.Finkelstein, G.Piccinelli}@cs.ucl.ac.uk
[2] Department of Computer Science, University of Hamburg, Germany
Lamersd@informatik.uni-hamburg.de
[3] IBM Software Group Germany and University of Stuttgart, Germany
LEY1@de.ibm.com
[4] IBM T. J. Watson Research Centre and University of Moratuwa, Sri Lanka
Sanjiva@us.ibm.com

Abstract. EOOWS (European workshop on Object Orientation and Web Services) brought together the academic and the industrial perspective on Web Services. Specific results presented by the workshop participants and the broad experience of the invited speakers provided the base for a lively discussion on the challenges and potentials of service-oriented computing in relation to object-oriented technologies and methodologies. This report summarises the activities and main threads of discussion within the workshop, as well as the conclusions reached by the participants.

1 Structure of the Workshop

The structure of the workshop was based on three main elements: presentations of technical papers, invited talks, and group discussions. The number of participants was around twenty, including presenting and non-presenting attendees.

The response to the call for papers had resulted in twenty-three submissions, of which only twelve could be selected for presentation. Despite the quality of all the submissions, time was a critical issue. The tension was between giving space to different contributions and ensuring for each presenter an adequate amount of time to communicate content and discuss feedback. Presenters were required to reduce general background and introductions to a bare minimum, and to concentrate on the core aspects of their work. Slide-sets were reviewed prior to presentation, and the amount of content balanced against the fifteen-minute time slot allocated to each presentation.

The two invited talks defined a more general context for the discussion of individual contributions. The first talk was given by Prof Mike Papazoglou (Tilburg University). Prof Papazoglou is the scientific coordinator of the forming European network of excellence in service-oriented computing. In his talk, Prof Papazoglou provided an architectural blueprint for web-service systems, as well as for development processes underpinning service engineering. Representing the

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 179–189, 2004.

academic view on Web Services, Prof Papazoglou also outlined some of the research challenges in areas such as planning, management, and quality of service. The second talk was given by Prof Frank Leymann (IBM Distinguished Engineer and member of the IBM Academy of Technology). Prof Leymann is one of the driving forces in the definition and standardization of Web Services. The talk provided valuable insight into the current role and the future developments of Web Services in the IT industry. In particular, Prof Leymann provided a comprehensive overview of the standardization activities currently ongoing for Web Services.

Discussion and network building developed throughout the workshop. As an example, at the end of each presentation the attendees were explicitly requested to identify themselves to the presenter if involved in ongoing or future activities with possible relations with the content of the presentation. A plenary discussion was held at the end of the workshop.

2 Themes and Objectives

Web Services are evolving beyond their SOAP, WSDL, and UDDI roots toward being able to solve significant real-world integration problems. Developers of Web Services systems are currently working on new generations systems that incorporate security, transactions, orchestration and choreography, GRID computing capabilities, business documents and processes, and simplified integration with existing middleware systems. Current economic issues continue to force consolidation and reduction in enterprise computing resources, which is resulting in developers discovering that Web Services can provide the foundation for the engineering and realisation of complex computing systems.

The question of how Web Services could and should change system and solution development is very much open. Are Web Services just about standards, or do they imply a new conceptual framework for engineering and development? Similarly open is the question of how requirements coming from system and solution development could and should make Web Services evolve. In particular, methodologies as well as technologies based on the object-oriented conceptual framework are an established reality. How do Web Services and object-orientation relate? How can Web Services leverage the experience built into current object-oriented practices?

The overall theme of the workshop was the relation between Web Services and object orientation. Such relation can be explored from different perspectives, ranging from system modelling and engineering to system development, management, maintenance, and evolution. Aspects of particular interest are the modularisation of a system into components and the (possibly cross-domain) composition and orchestration of different modules. Components and composition are closely connected with the issue of reuse, and an important thread of discussion encouraged within the workshop addressed the way in which Web Services impact reuse.

The objective of the workshop was twofold: assessing the current work on Web Services, and discussing lines of development and possible cooperation. Current work included research activities as well as practical experiences. The assessment covered an analysis of driving factors and a retrospective on lessons learned. The identification and prioritisation of new lines of research and activity was a key outcome of the workshop. In particular, the intention was to foster future cooperation among the participants.

3 Main Lines of Discussion

The discussion activity during the workshop was substantially based on the material presented by the participants. The various contributions are briefly summarized in Appendix A. The papers related to each contribution were published in the IBM Research Reports series (RA220 – Computer Science).

The main lines of discussion within the workshop are particularly well represented by the contributions from João Paulo A. Almeida, Luís Ferreira Pires, and Marten J. van Sinderen (on seamless interoperability) [Appendix A.1], and Vincenzo D'Andrea and Marco Aiello (on the relation between services and objects) [Appendix A.4]. The reminder of this section provides extracts from the related documents.

3.1 Seamless Interoperability

In order to enable the cooperation of distributed applications, Web Services must accommodate the heterogeneity of middleware platforms, programming languages and other technologies in which such applications are realized. Ideally, application developers should be shielded from the existence of different middleware platforms and programming language abstractions, manipulating a set of consistent high-level constructs to access both the services that are located within the same technology domain and services that are implemented in other technology domains.

The success of Web Services will depend on their consistent adoption in intra-domain development. This means that the abstractions used in Web Services languages and protocols should match as closely as possible with the abstractions used for development of applications. If Web Services are confined to inter-domain interoperation, abstractions manipulated by intra-domain middleware platforms will indeed diverge from abstractions manipulated across technology domains, and there will always be a"seam" between the abstractions manipulated in a technology domain and abstractions used in inter-domain interoperation.

The lack of seamless interoperation can be observed in different attempts to provide mappings between Web Services abstractions and abstractions supported by different middleware platforms, such as, e.g., the mappings from and to Java in the JAX-RPC specification [11], the mappings from and to .NET's Common Type System [4] and the upcoming mappings from and to CORBA IDL [6, 7].

These mappings are not sufficient to overcome the intrinsic conceptual differences of the abstractions adopted. For example, a Java developer that is used to passing remote object references as parameters in J2EE [12] is not able to do so if an object is to be exposed as a Web Service endpoint [11]. This is because the concept of remote object references is not directly supported in a standardized way in SOAP and WSDL, and hence this abstraction has no direct counterpart. Several other examples of mismatch can be identified when considering these mappings, in terms of fault semantics, type mappings, etc. This is a recurring pattern that we have seen earlier in the development of mappings to and from OMG Interface Definition Language (IDL) to Java, C, C++, Ada, Smalltalk, etc. [5].

Abstractions of particular domains are not the only obstacles for seamless interoperation. For applications to achieve meaningful interaction, they must agree on the application protocols to use. These protocols are referred to as *application choreographies* [10] in the context of Web Services, and relate to the behavioural or dynamic aspects of an application. Behaviour complements static aspects of a system, such as interface signatures, data structures and deployment descriptors. Divergences in the behaviour of components of different technology domains offer challenges to transparent inter-domain interoperability. Emerging Web Service standards such as BPEL [1] might prove extremely valuable for choreography.

3.2 Services and Objects

Is a service an object? We propose a negative answer to this question, but many similarities connect OOP (object-oriented programming) and SOC (service-oriented computing). More object related concepts should move into the service-oriented world in order to enhance the technology and, perhaps, clarify the role and scope of web services. Here are the most immediate examples of concept migration:

Inheritance. Interface inheritance seems to be the most immediate to apply to Web Services. Let us consider a payment service P, and a sub-type Pr of P that also supports the capability to acknowledge receipt. In a workflow, P could be substituted by Pr. Interface inheritance would guarantee that the same port types of P are implemented in the sub-typed service Pr. Inheritance enables service substitution, service composition, and it induces a notion of inheritance on entire compositions of services. Let us consider a workflow W built on the service P and second workflow W1 with the same data and control links as W, but built on the service P1. Can we say that W1 inherits from W or that W1 is a specialization of W1?

Polymorphism. Both inclusion polymorphism and overloading can be extended to the service paradigm. A composition operation in a workflow may have different meanings depending on the type of the composed services. For example, composing a payment and a delivery service may have a semantics for which the two services run in parallel; on the other hand, the composition of two sub-typed services in which the payment must be acknowledged by the payers

bank and the delivery must include the payment transaction identifier have the semantics of a sequencing the execution of the services.

Composition. A formal and accepted notion of composition is currently missing in the SOC domain, but inheritance and polymorphism could induce such a notion as composition over services. Could this help dissolve the fog around the meaning of composition for web services? Could this bring together "syntacticians", which claim that nothing can be composed if not by design, with "semanticians", which claim that anything can be composed automatically? Perhaps not, but it could fill some of the gaps left by standards which could lend themselves to different semantic interpretations. A noticeable example is BPEL [1], which is emerging as the standard for expressing aggregations of web services.

State-fullness. The difference between stateless class definitions and state-full objects in OOP can impact Web Services technology, where services are stateless entities. Web Services definitions (e.g. in WSDL) seem close to class definitions, but the notion of state is paramount. Software entities need to access each other's state in order to fully interoperate. Here the parallel is with the history of HTML pages. When first introduced, HTTP/HTML interactions were stateless. Still, the limitations to client-server communication called for the introduction of state-full solutions, which were addressed by 'cookies' [2].

The object-oriented paradigm has a solid formal background and is a well-established reality in today's computer science. Service-oriented computing is, on the other hand, a new emerging field, which tries to realize global interoperability between independent services. To meet this goal, service-oriented technology will need to solve a number of challenging issues. Precise service semantics is a critical issue, and OO body of knowledge could provide a solid base on which to build.

4 Conclusions

Web Services are one of the hottest topics in the IT industry at the moment. Their success factor is the fact that their sole focus is on leveraging existing business resources. Companies have been accumulating all sorts of technology. Web Services are about providing a much needed integration infrastructure.

So far Web Services have been used within the boundaries of the enterprise. Basically, existing systems and application are being coated with a Web Service layer. It is not ideal, but it already provides huge benefits to system integrators. This trend will probably continue for at least the coming 18/24 months. The next step will be to actually re-engineer old systems and engineer new system based on service-oriented principles.

Fundamental issue for Web Services is standardisation. Major IT players such as IBM, Microsoft, BEA, and SAP are cooperating with organisations such as Oasis and W3C towards the definition of key standards in areas such as security and business processes. Other major issues concern systems engineering. Traditional approaches (e.g. object orientation) provide a valuable starting point, but specific models and methodologies are required.

Once standards are in place, the main challenge will be to develop a comprehensive body of systems engineering knowledge that can be concretely absorbed by system developers. In particular, service-based models require a close coupling between business engineering and IT. Specific methodologies are needed to effectively bridge the gap across the two domains.

The message for technical people is: if you are in the IT industry and you are not aware of Web Services ... you are already late. Catch up ASAP! If you are in research, don't bother with the basic tools and technology ... the industry is already working on them. Concentrate on the 2 to 3 year time horizon.

References

1. BEA, IBM, Microsoft, SAP AG, and Siebel. Business Process Execution Language for Web Services, 2003. Available at http://www-106.ibm.com/developerworks/library/wsbpel
2. Kristol D. HTTP cookies: Standards, privacy, and politics. ACM Transactions on Internet Technology (TOIT), 1(2):151-198, 2001.
3. Marton A., Piccinelli G., and Turfin C. Service Provision and Composition in Virtual Business Communities. Symposium on Reliable Distributed Systems 1999: 336-341
4. Microsoft Corporation. .NET Development. Available at http://msdn.microsoft.com/library/ /en-us/dnanchor/html/netdevanchor.asp
5. Object Management Group. Catalog of OMG IDL / Language Mappings Specifications. Available at http://www.omg.org/technology/documents/idl2x_spec_catalog.htm
6. Object Management Group. CORBA-WSDL/SOAP specification, ptc/03-01-14, Jan. 2003.
7. Object Management Group. Joint Revised Submission to the WSDL-SOAP to CORBA Interworking RFP, mars/03-03-03, March 2003.
8. Piccinelli G., Emmerich W., and Finkelstein F. Mapping Service Components to EJB Business Objects. EDOC 2001: 169-173
9. Piccinelli G., Emmerich W., Zirpins C., and Schütt K. Web Service Interfaces for Inter-Organisational Business Processes: An Infrastructure for Automated Reconciliation. EDOC 2002: 285-292
10. Schmidt, D. and Vinoski, S. Object Interconnections: CORBA and XML – Part 3: SOAP and Web Services, C/C++ Users Journal C++ Experts Forum, Sept 2001.
11. Sun Microsystems. Java API for XML-Based RPC Specification 1.0, June 2002.
12. Sun Microsystems. Java 2 Platform Enterprise Edition. Available at http://java.sun.com/j2ee
13. The Parlay Group. Parlay Web Services Architecture Comparison, October 2002. Available at http://www.parlay.org/specs/ParlayWebServices-ArchitectureComparison1_0.pdf
14. Universal Description, Discovery and Integration (UDDI) project. UDDI: Specifications. Available at http://www.uddi.org/specification.html

Appendix A. Selected Contributions

The full version of the papers was published in the IBM Research Report RA220 Computer Science. The papers are also available online at:
http://www.cs.ucl.ac.uk/staff/g.piccinelli/eoows/eoows-programme.htm

A.1 Web Services and Seamless Interoperability

João Paulo A. Almeida, Luís Ferreira Pires, Marten J. van Sinderen
Centre for Telematics and Information Technology, University of Twente, The Netherlands

Web Services technologies are often proposed as a means to integrate applications that are developed in different middleware platforms and implementation environments. Ideally, application developers and integrators should be shielded from the existence of different middleware platforms and programming language abstractions. This characterizes seamless interoperability, in which a set of consistent constructs is manipulated to integrate both the applications or services that are located both in the same and in different technology domains. In this paper, we argue that Web Services are not sufficient to facilitate seamless interoperability. We also outline some developments that may be used in a systematic approach to seamless interoperability within the context of the Model-Driven Architecture.

A.2 enTish: An Approach to Service Description and Composition

Stanislaw Ambroszkiewicz
Institute of Informatics, University of Podlasie, Poland

A technology for service description and composition in open and distributed environment is proposed. The technology consists of description language (called Entish) and composition protocol called entish 1.0. They are based on software agent paradigm. The description language is the contents language of the messages that are exchanged (between agents and services) according to the composition protocol. The syntax of the language as well as the message format are expressed in XML. The language and the protocol are merely specifications.

To prove that the technology does work, the prototype implementation is provided available for use and evaluation via web interfaces starting with www.-ipipan.waw.pl/mas/ . Related work was done by WSDL + BPEL4WS + (WS-Coordination) + (WS-Transactions), WSCI, BPML, DAML-S, SWORD, XSRL, and SELF-SERV. Our technology is based on similar principles as XSRL, however the proposed solution is different. The language Entish is fully declarative. A task (expressed in Entish) describes the desired static situation to be realized by the composition protocol.

A.3 Modularizing Web Services Management with AOP

María Agustina Cibrán, Bart Verheecke
System and Software Engineering Lab, Brussels, Europe

Web service technologies accelerate application development by allowing the selection and integration of third-party Web Services, achieving high modularity, flexibility and configurability. However, current approaches to integrate Web Services in client applications do not provide any management support, which is fundamental for achieving robustness. In this paper we show how Aspect Oriented Programming (AOP) can be used to modularise service management issues in service-oriented applications. To deal with the dynamic nature of the service environment we suggest the use of a dynamic aspect oriented programming language called JAsCo. We encapsulate the management code in aspects placed in an intermediate layer in between the application and the world of Web Services, called Web Services Management Layer (WSML).

A.4 Services and Objects: Open Issues

Vincenzo D'Andrea and Marco Aiello
Department of Information and Telecommunication Technologies, University of Trento, Italy

One of the common metaphors used in textbooks on Object-Oriented programming (OOP) is to view objects in terms of the services they provide, describing them in "service oriented" terms. This opens a number of interesting questions, moving away from the simple view of OOP as an implementation tool for Web Services. First of all: if an Object is a Service, can we also say that a Service is an Object? While the short answer seems to be negative, there are several connections between the two concepts and it is possible to exploit the large repository of methodological tools available in OOP. What are the counterparts, in terms of services, of concepts like class or instance? Is it possible to apply techniques as containment or inheritance to services? What are interfaces, properties and methods for services? In this paper we try to start building some connections, underlining the open issues and the grey areas.

A.5 UML Modelling of Automated Business Processes with a Mapping to BPEL4WS

Tracy Gardner
IBM UK Laboratories, Hursley Park, UK

The Business Process Execution Language for Web Services (BPEL4WS) provides an XML notation and semantics for specifying business process behaviour based on Web Services. A BPEL4WS process is defined in terms of its interactions with partners. A partner may provide services to the process, require

services from the process, or participate in a two-way interaction with the process.

The Unified Modelling Language (UML) is a language, with a visual notation, for modelling software systems. The UML is an OMG standard and is widely supported by tools. UML can be customized for use in a particular modelling context through a 'UML profile'. We describe a UML Profile for Automated Business Processes that allows BPEL4WS processes to be modelled using an existing UML tool. We also describe a mapping to BPEL4WS that can be automated to generate Web Services artefacts (BPEL, WSDL, XSD) from a UML model meeting the profile.

A.6 A Classification Framework for Approaches and Methodologies to Make Web Services Compositions Reliable

Muhammad F. Kaleem
Technical University Hamburg-Harburg

Individual Web Services can be composed together to form composite services representing business process workflows. The value of such workflows is directly influenced by the reliability of the composite services. There is considerable research concerned with reliability of Web Services compositions. There is, however, no clear definition of reliability that establishes its scope in the context of a composite service. We propose a definition of composite service reliability in this paper that takes into account different aspects affecting reliability of a composite service. We also put this definition to use as the basis of a framework that can be used for classification of approaches and associated methodologies for making composite services reliable. The framework can be used to identify which aspect of composite service reliability is addressed by a particular approach. We will also reference some selected methodologies to classify them according to the aspect of reliability they address. A definition of composite service reliability and its use to classify methodologies for composite service reliability as described in this paper will prove useful for comparing and evaluating methodologies for making composite services reliable, and will also have a bearing on the quality of service aspects of architectural styles and methodologies of software solutions based on Web Services compositions.

A.7 Requestor Friendly Web Services

Ravi Konuru and Nirmal Mukhi
IBM Research, USA

Web service providers rely on the Web Service Description Language (WSDL) as the way to communicate information about an available service to a service requestor. This description or meta-data of the service is used by a service requestor to inspect the available interfaces and to access the service. In this paper, we argue that publishing a WSDL with a functional description of a service

alone is not requestor friendly, i.e., it does not allow the requestor any flexibility in improving the end-to-end responsiveness and customize the Web Service behaviour. We present some scenarios to back this argument and also outline a spectrum of solution approaches.

A.8 Using Web Services in the European Grid of Solar Observations (EGSO)

Simon Martin and Dave Pike
Space Science and Technology Department, Rutherford Appleton Laboratory, UK

The European Grid of Solar Observations (EGSO) is employing Grid computing concepts to federate heterogeneous solar data archives into a single 'virtual' archive, allowing scientists to easily locate and retrieve particular data sets from multiple sources. EGSO will also offer facilities for the processing of data within the Grid, reducing the volume of data to be transferred to the user. In this paper, we examine the use of Web Services in EGSO as a means of communicating between the various roles in the system.

A.9 A Web Services Based System for Data Grid

Irene Pompili, Claudio Zunino, and Andrea Sanna
Politecnico di Torino, Italy

In this paper an architecture for a data brokerage service will be proposed. The brokerage service is a part of the system that is being implemented within the European Grid for Solar Observations (EGSO) to provide a high-performance infrastructure for solar applications. A broker interacts with providers and consumers in order to build a profile of both parties. In particular, the broker interacts with providers in order to gather information on the data potentially available to consumers, and with the consumers in order to identify the set of providers that are most likely to satisfy specific data needs.

A.10 Agile Modelling and Design of Service-Oriented Component Architecture

Zoran Stojanovic, Ajantha Dahanayake, Henk Sol
Delft University of Technology, The Netherlands

Component-Based Development (CBD) and Web Services (WS) have been proposed as ways of building high quality and flexible enterprise-scale e-business solutions that fulfil business goals within a short time-to-market. However, current achievements in these areas at the level of modelling and design are much behind the technology ones. This paper presents how component-based modelling

and design principles can be used as a basis for modelling a Service-Oriented Architecture (SOA). Proposed design approach is basically model-driven, but incorporates several agile development principles and practices that provide its flexibility and agility in today's ever-changing business and IT environments.

A.11 A Blueprint of Service Engineering

Christian Zirpins, Toby Baier, Winfried Lamersdorf
University of Hamburg, Germany

The rationale behind service oriented computing is to lift inter-organisational integration on a higher level of effectiveness and efficiency. E-services promise to offer means for floating modularisation of arbitrary organisational assets into components that can be dynamically offered, discovered, negotiated, accessed and composed in an open environment. From a technical point of view electronic services are software systems that have to be implemented on top of conventional information and communication technology. As an important step into that direction, the Web Service architecture has laid the foundation for interoperable communication between arbitrary systems. This extended abstract outlines an approach to plan, build and run application-level services on top. Therefore, a fundamental notion of service, originating from distributed systems, is being extended by a specific concept of cooperative interaction processes. Accordingly, an application-level service model and corresponding service engineering mechanisms are proposed that are being realised as middleware based on OGSA Web Services and BPEL4WS processes.

Advancing the State of the Art in Run-Time Inspection

Robert Filman[1], Katharina Mehner[2], and Michael Haupt[3]

[1] NASA Ames Research Center, USA rfilman@arc.nasa.gov
[2] Universitaet Paderborn, Germany mehner@upb.de
[3] Technische Universität Darmstadt, Germany haupt@informatik.tu-darmstadt.de

1 Introduction

Modern software development is inconceivable without tools to inspect running programs. Run-time inspection is a crucial factor for both building complex systems and for maintaining legacy systems. Run-time inspection includes not only querying of program state but also controlling its execution. Applications of run-time inspection range from code-level tasks like debugging, profiling, tracing, testing, and monitoring to conceptual activities such as program comprehension, software visualization and reverse engineering. New applications incorporate run-time inspection as a programming concept in the style of event-condition-action rules.

The diversity of programming paradigms such as configurable software, components, aspect-orientation, generative programming, real-time programming, distributed programming, ubiquitous computing, applets and web services emphasizes the need to deal with heterogeneous run-time information and different levels of abstraction. Lacking well-established technologies and models for representing and accessing program dynamics, tools must use ad-hoc mechanisms. This limits reuse and interoperability. De facto standards for run-time inspection such as the Java Platform Debugger Architecture (JPDA) have improved the situation but do not cope with all requirements. Implementers seeking to create debugging environments for ubiquitous computing are faced with even greater difficulties.

The workshop sought to identify best practices and common requirements, to specify conceptual data, control models and implementation approaches for run-time inspection and to discuss practical issues such as standardized APIs and data exchange formats.

The contributions for the workshop covered a wide range of topics. We divided the contributions into two sessions: classical application of run-time inspection and novel architectures made possible by the expanded use of run-time inspection. Classical uses of run-time inspection included debugging and program visualization. The dominant novel use of run-time inspection was as a foundation for dynamic aspect-oriented programming.

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 190–196, 2004.

2 Session on Run-Time Inspection: Methods, Applications, and Special Issues

The first three position papers of the first session presented run-time tools: a debugger and two visualization tools, one of which also allows run-time program manipulation. The last two position papers tackled specific problems in run-time inspection: data collection and scalability.

E. Tanter, P. Ebraert. A Flexible Approach to Interactive Runtime Inspection

Tanter and Ebraert propose using *behavioral reflection* as a foundation for run-time inspection. Behavioral reflection allows introspection (querying the parts of a program describing its dynamic behavior), and intercession (controlling its execution). Tanter presented a tool for run-time inspection which is based on a previously developed system called *Reflex*.

Reflex supports partial (behavioral) reflection for Java by allowing the programmer to choose which operations of which classes or objects should be reified during which part of the program lifetime. These reifications of base-level occurrences of operations are accessible statically and at run-time in terms of meta-objects. Reflex is implemented using the Java Platform Debugger Architecture.

This run-time inspection tool provides a graphical interface representing the meta-objects. Through this interface the program can be manipulated at run-time. Tanter pointed out that the particular challenge of such a tool lies in dealing with the visual load and in synchronizing the user interaction with the running program by providing direct feedback. He also envisioned raising the level of abstraction to yet another meta-level by viewing the meta-objects as units of separated concerns and by providing an additional meta layer for monitoring and manipulating these units.

This work is related to the position papers on dynamic AOP discussed in the next section. The proposal of behavioral reflection as a foundation of run-time inspection was picked up in the discussion group on a model for run-time inspection.

B. Lewis. Recording Events to Analyze Programs

Lewis presented the *Omniscient Debugger*, a Java debugger which allows making a detailed recording (also called trace) of a running Java program and stepping through this recording, going both backwards and forwards. After each such step, a detailed state of the program is presented. This approach allows an intensive examination of the execution history without ever making a choice where to set a breakpoint. Going backwards is extremely useful in tracing problems to the place where they first occurred—the preeminent question is debugging is "How did that happen?"

Lewis entertained the group with a plush-toy demonstration of the importance of such a debugger: bugs are not "bugs" but different animals, namely snakes and lizards who hide in the grass. Snake and lizard tails often stick out of the grass. Following the bug to its source means grabbing the animal by its tail. If the program produces an incorrect answer you have a tail on which you can pull. If you pull on a snake's tail long enough you will get to its head; by analogy, this is finding the source of the bug. The Omniscient Debugger facilitates this through its ability to move backwards in time, like moving along the body of the snake. However, if you pull on a lizard's tail long enough it will eventually break, keeping you from getting to the source of the bug. Lizard-like bugs are endemic to break-point centered debuggers, because you can't pull the tail much at all—you know the "now," but not the "then."

Lewis pointed out that even with the arbitrary reversibility of the omniscient debugger, some bugs are still intractable. Bugs due to the failure to do something cannot be traced back to a mistaken action. If a program fails to produce a correct answer then one does not have an animal tail to grab. However, the Omniscient Debugger can be used to help search for the hidden-in-the-grass tail.

The Omniscient Debugger is implemented by Java byte code instrumentation, and relies heavily on timestamps. An instrumented event causes an average slowdown of two microseconds. At best, a slowdown of factor two can be achieved for an entire program. Due to the efficient implementation and clever recognition of state-preserving routines, the Omniscient Debugger can work with programs generating a ten million events.

A. Cain, J. Schneider, D. Grant, T. Chen. Runtime Data Analysis for Java Programs

Cain et al. examined the usefulness of the Java Platform Debugger Architecture (JPDA) for data flow analysis. Data flow analysis requires capturing the definition and the references of all variables in a program. Often, it is desirable that the scope of the analysis can be targeted. Because local variables and array elements are not covered by the JPDA, Cain presented a hybrid solution using source code transformation and run-time analysis through the JPDA.

A. Zaidman, S. Demeyer. Program Comprehension Through Dynamic Analysis

Zaidman and Demeyer addressed the problem of the size of trace data. Huge traces are not only a problem of memory but also a cognitive problem for the user. Their solution to reducing trace size is to compress trace data. They propose to cluster method-call events based on the same frequency of occurrence in a program run. This clustering technique is based on the heuristic assumption that methods working together to reach a common goal are be executed about the same number of times. Zaidman suggested that the spectral visualizations of these clusters could also be used to do dissimilarity measures and to search for visual patterns.

H. Leroux, C. Mingins, A. Requile-Romanczuk. JACOT: A UML-Based Tool for the Runtime-Inspection of Concurrent Java Programs

Leroux et al. presented a tool called *JACOT* to dynamically visualize a running program by means of animated UML sequence diagrams. A sequence diagram can show object interaction over time and thus can present the history of program execution. JACOT has a special focus on threading and exceptions. It provides an animated state chart for depicting thread state and an activity diagram for depicting exception flow. The tool is implemented using the Java Platform Debugger Architecture.

Session Conclusion

The discussion group on visualization noted that the presentations by Lewis and Leroux contrasted different visions of how to present the history of program execution. Lewis's approach provides a detailed, step-by-step history. Leroux presents a quickly understood though less-precisely meaningful visual abstraction. Clearly, the ultimate run-time inspection tool would present both kinds of mechanisms, and allow the user to switch between them as needed.

3 Session on Dynamic Aspect-Oriented Programming

The presenters in this session focused on implementation approaches to dynamic Aspect-Oriented Programming (AOP) systems and language design problems. Implementation languages used in the papers were Common LISP, Java, and Smalltalk.

P. Costanza. Dynamically Scoped Functions for Runtime Modification

Costanza argued that a function definition is dynamically scoped if its binding is *always* looked up in the current call stack, regardless of which point in program execution is currently being processed. This contrasted with lexically scoped definitions, whose meaning is determined by examining a program's syntactic structure.

Costanza claimed that dynamically scoped functions are useful for modifying an application's behavior, a common task for aspect oriented programming. The paper proposes extending LISP with dynamically scoped functions to be able to decorate functions with additional enclosing behavior.

S. Chiba, Y. Sato, M. Tatsubori. Using HotSwap for Implementing Dynamic AOP Systems

Chiba et al. presented *Wool*, an implementation of dynamic aspect weaving for Java utilizing the standard JVM's HotSwap capabilities. HotSwap allows for

changing class implementations at run-time via a function that is part of the Java debugger API.

Wool follows a hybrid approach in decorating join points with additional behavior at run-time. At first, all active join points are registered as debugger break points. The system checks each time such a break point is reached to see if any aspects should be invoked. However, there is a large execution costs to this repeated checking. When a break point has been reached often enough (decided by a simple heuristic), the corresponding method is replaced with a modified version that directly calls the aspect.

S. Aussmann, M. Haupt. Axon — Dynamic AOP Through Runtime Inspection and Monitoring

Haupt and Aussmann presented Axon, an implementation approach to dynamic AOP that relies solely on the JVM's debugger to intercept application execution at join points and branch to aspect functionality.

Axon's underlying model is strongly influenced by the concept of event-condition-action (ECA) rules found in active databases. An ECA rule comprises an event describing a (complex) situation in the database, a condition that is evaluated when the event is signaled and an action that is executed when the condition evaluates to true.

Haupt argued that ECA rules and aspects are related in that join points and pointcuts, marking that a certain point in execution has been reached, form an event algebra and advice correspond to actions. From the observation of this relationship, Axon's aspect model was developed that strongly decouples the different parts of an aspect from each other.

S. Hanenberg, R. Hirschfeld, R. Unland, K. Kawamura. Aspect Weaving: Using the Base Language's Introspective Facilities to Determine Join Points

Using a sample implementation of a generic observer in AspectJ, Hanenberg et al. first showed that most existing AOP languages suffer from not being thoroughly equipped with reflective capabilities. An observer ensures that any change to a subject's instance variables that is done through the subject's interface is reported to the observers. However, this is not true if the instance variables are changed by some other means, e. g., if they are instances of reference types and a modification of their own data is done.

Having presented this problem, the authors demonstrated a solution based on the using the reflective capabilities of Smalltalk and the facilities of the AspectS AOP extension to Smalltalk. In their solution, reflection is used to traverse an observed object structure and to decorate all necessary places—i. e., all places where data belonging to the structure may be changed—with notification calls, thereby ensuring that *any* change in the subject is reported to the observers.

Session Conclusion

The two concrete implementation approaches presented by Chiba et al. and Aussmann/Haupt show that implementing run-time AOP systems is an active area. Both approaches utilize an interception-based strategy, illustrating the relationship between run-time AOP and event-based systems. Using dynamically scoped functions to implement dynamic AOP contributes to a better understanding of the semantics of such systems.

The work of Hanenberg et al. shows that purely lexical join points are not enough to precisely express the interactions of aspects with the applications they decorate. There is a need for mechanisms that are able to capture and express logical requirements of application structures.

4 Discussion Topics of Break-Out Groups

The workshop broke into three groups to discuss particular topics in greater detail.

4.1 A Model for Run-Time Inspection

The aim of this group was to describe a model capturing the essence of run-time inspection.

The group argued that the term "inspection" is a source of misunderstanding. It was coined by the workshop organizers in analogy to the term guided inspection, sometimes also called code inspection, which refers to a testing practice using code walkthroughs. In analogy to guided inspection, run-time inspection can be seen as being purely observational, perhaps also including the possibility of pausing program execution. However, this definition does not capture the ability to change the program's execution. Some of the participants considered this to be an essential part of run-time inspection. Reflection was therefore proposed as a foundation for run-time inspection because it includes the possibility of making changes to the program and its execution.

The model for run-time inspection proposed by this discussion group is built not only on reflection but also draws on existing models from the related area of software visualization. Models for software visualization typically distinguish the phases data collection, data transformation, and data presentation. The following model aims at clearly identifying the different phases in the process of run-time inspection:

- Running the program
- Collecting data
- Transforming data
- Presenting data
- Effecting changes based on presentation of data
- Propagate changes to the program

This is a cyclic process. The changes become effective in the running program.

4.2 Visualization

This group was considered the visual presentation of information about running programs. The group noted that the first step in visualization is always defining what information is to be visualized. The group then addressed the problem of dealing with high volumes of information gathered at run-time. Suggestions to shape a call graph to one line per function were contrasted with animation of a class diagram by highlighting fields and members involved in method invocation during (re)play. Information compression was addressed through clustering of method calls based on the coupling, i. e., the number of calls between two methods. Also, different diagrammatic notations were discussed such as tree-like structures for hierarchical information and spiral and kiviat (spider) structures.

4.3 Applications

This group was looking at applications of run-time inspection with a primary focus on applications for dynamic aspect-oriented programming. Client-reside code and application servers were identified as promising application areas for technologies that can make changes to running code. Examples of uses of such technologies in these domains include:

- Bug fixes
- Usage monitoring
- Integration with other applications

The group raised the important issue of the safety of run-time changes. Safety concerns arise both for security and for system integrity—it is easy to imagine making dynamic changes that corrupt a system's logical state. After all, even in tightly release-controlled environments, a high percentage of bug fixes themselves introduce new bugs. This issue reflects similar concerns in AOP, which has the hope of modularizing changes in aspects but has long recognized the potential problem of conflicting aspects.

Aliasing, Confinement, and Ownership in Object-Oriented Programming

Dave Clarke[1], Sophia Drossopoulou[2], and James Noble[3]

[1] Utrecht University, The Netherlands. dave@cs.uu.nl
[2] Imperial College, London, UK. sd@doc.ic.ac.uk
[3] Victoria University of Wellington, New Zealand. kjx@mcs.vuw.ac.uk

Abstract. The power of objects lies in the flexibility of their interconnection structure. But this flexibility comes at a cost. Because an object can be modified via any alias, object-oriented programs are hard to understand, maintain, and analyse. Aliasing makes objects depend on their environment in unpredictable ways, breaking the encapsulation necessary for reliable software components, thus making it difficult to reason about and optimise programs, obscuring the flow of information between objects, and introducing security problems.
Aliasing is a fundamental difficulty, but we accept its presence. Instead we seek techniques for describing, reasoning about, restricting, analysing, and preventing the connections between objects and the flow of information between them.

1 Introduction

The aim of IWACO was to address the following issues:

- models, type and other formal systems, programming language mechanisms, analysis and design techniques, patterns and notations for expressing object ownership, aliasing, confinement, uniqueness, or information flow.
- optimisation techniques, analysis algorithms, libraries, applications, and novel approaches exploiting object ownership, aliasing, confinement, uniqueness, or information flow.
- empirical studies of programs or experience reports from programming systems designed with these issues in mind.
- novel applications of aliasing management techniques such as ownership types, confined types, region types, and uniqueness.

More poetically, we ask how do we take arms against the sea of objects?

There were 9 papers submitted to the workshop, of which 8 were accepted. An informal proceedings was handed out at the workshop [Cla03]. About 25 people registered for the workshop beforehand, though 32 people were in attendance. Many of the leading researchers in the field were present. The workshop was organised into a series of talks, including one invited talk, with plenty of time interleaved for discussion. More discussion time was scheduled towards the end of the day. This worked very well. We concluded the day with a group dinner in a local beer hall, at which much of the discussion continued.

F. Buschmann et al. (Eds.): ECOOP 2003 Workshop Reader, LNCS 3013, pp. 197–207, 2004.

1.1 History

The issues addressed in this workshop were first brought into focus with the *Geneva Convention on the Treatment of Object Aliasing* [HLW+92]. The *Intercontinental Workshop on Aliasing in Object-oriented Systems (IWAOOS)* at ECOOP'99 addressed similar issues and can be seen as a precursor to our workshop.

2 Invited Talk

The workshop began with the invited talk by Peter O'Hearn, from Queen Mary, University of London. Peter is not a member of the object-oriented community, as such, but has been working on new logics — separation logics — for reasoning about low level, pointer-based programs. We invited Peter to give a talk because we felt that that the our community needed to know about his work. Inviting Peter turned out to be a fortuitous decision, as he was able to provoke quite some discussion with his often deliberately controversial remarks.

Separation and Information Hiding
Peter O'Hearn

Separation logic [IO01] is based on a logical connective which enables one to express properties of disjoint parts of a store, even in the presence dangling pointers in the partitions. Separation logic has seen much success in specifying low level pointer programs in a concise manner.

Apart from giving an example-driven overview of separation logic, in his talk Peter described proof rules for information hiding using separation logic. The separating conjunction can be used to partition the internal resources of a module from those accessed by the module's clients. The use of a logical connective gives rise to a form of dynamic partitioning, where it is possible track the transfer of ownership of portions of heap storage between program components. The beauty is that the logic enables one to enforce the separation even in the presence of mutable data structures with embedded addresses that may be aliased.

Comments. Peter's talk, though not dealing specifically with object-oriented constructs, certainly gave many members of the audience a lot to think about: logics are possible which incorporate explicitly notions of aliasing and enable local reasoning in its presence. Lacking, however, was any real indication of how to deal with inheritance, subtyping, nor achieving modular soundness of specifications. In spite of the fact that Peter's talk was interesting and his ideas showed promise, Doug Lea made the point that the formalism was too heavy for most programmers to use. Indeed this tension between expressiveness and what programmers would put up with was a recurring theme throughout the day.

3 The Presentations

Safe Runtime Downcasts with Ownership Types
Chandrasekhar Boyapati, Robert Lee, Martin Rinard

This paper described an efficient technique for supporting safe runtime downcasts in a system with ownership types. This technique uses type-passing, but avoids the associated significant space overhead by storing only the runtime ownership information that is potentially needed to support safe downcasts. As the technique does not use any inter-procedural analysis, it preserves the separate compilation model of Java. The technique has been implemented in the context of Safe Concurrent Java, which is an extension to Java that uses ownership types to statically guarantee the absense of data races and deadlocks. The approach is JVM compatible: the implementation translates programs to bytecodes which can be run on regular JVMs.

Comments. Chandra's talk gave more of an overview of the applications of ownership types [CPN98], rather than talk about the specific topic matter at hand. This was perhaps an important introduction to ownership types for the few people in the audience not familiar with them.

Regarding the matter in the paper, it was noted that the additional ownership information carried around to enable downcasts was not made available to the programmer. Doug Lea suggested that it should be, as programmers would no doubt find some use for it.

Cheaper Reasoning with Ownership Types
Matthew Smith, Sophia Drossopoulou

The paper described the use of ownership types in facilitating program verification. In particular, the claim is that an assertion which is established for some part of the heap which is unaffected by some execution will still hold after this execution. Ownership and effects are extended to assertions to enable reasoning about which executions do not affect which assertions. The ideas are described with the use of an example, though a formal system is also outlined.

Comments. Although preliminary, Matthew's talk presented some interesting ideas using effects systems based on ownership types. What remains to do is to focus on the non-aliasing guarantees provided by ownership types. Discussion continued offline with Peter Müller and Arnd Poetzsch-Heffter, who both have experience in using an ownership type system for reasoning about Java programs.

Formalising Eiffel References and Expanded Types in PVS
Richard Paige, Jonathan Ostroff, Phillip Brooke

Ongoing work was described in which a theory of Eiffel reference and expanded (composite) types is formalised. The theory was expressed in the PVS specification language, thus enabling the use of the PVS theorem prover and model

checker to semi-automatically prove properties about Eiffel structures and programs. The theory is being used as the basis for automated support for the Eiffel Refinement Calculus.

Comments. This talk provided some insights into the large scale industrial application of program verification. The talk focussed on one particular technique for managing aliasing in such applications. The audience was interested in seeing how applicable the suggested technique would be.

Connecting Effects and Uniqueness with Adoption
John Tang Boyland

In previous work, John had discussed how the concepts of uniqueness and effects were interdependent. This work shows how Fähndrich and DeLine's "Adoption and Focus" [FD02], a proposal for handling linear pointers in shared variables, can be extended to connect the two concepts. The innovations presented in the paper include the ability to define adoption relations between individual fields, rather than whole objects, and the ability to "focus" on more than one adoptee at a time. The resulting system uses recursive alias types, "permission closures" and "conditional permissions." The paper then goes on to demonstrate how proposed effect and uniqueness annotations can be represented in the type system.

Comments. Although this work is in its infancy, John managed to outline what may be a useful connection: that adoption connects effects and uniqueness. The work is a rather non-standard synthesis of many of techniques which are being pushed around at the current moment. It would be informative to see the simplest system in which this connection can be elucidated, as the present system is rather complex.

Heap Monotonic Typestates
Manuel Fähndrich, K. Rustan M. Leino

The paper defines the class of heap monotonic typestates. The monotonicity of such typestates enables sound checking algorithms without the need for non-aliasing regimes of pointers. The basic idea is that data structures evolve over time in a manner that only makes their representation invariants grow stronger, never weaker. This assumption guarantees that existing object references with particular typestates remain valid in all program futures, while still allowing objects to attain stronger typestates. The system is powerful enough to establish properties of circular data structures.

Comments. Typestate [SY86] is a notion which was introduced in the 80's to capture the idea that entities can be in different states and only certain operations can be performed when in these states. By putting these into the type system, stronger checking becomes possible. Unfortunately, this is very difficult in the presence of aliasing. Manuel and Rustan demonstrate that for a certain

natural class of typestate the problem becomes much simpler. Their technique works irrespectively of aliasing (so long as you can keep a reasonable hold on the aliasing). This work (further supported by the opinions of its authors) provides evidence that aliasing should never be the key focus; one must consider foremost what properties of the object at the end of a reference can be determined.

Towards a Model of Encapsulation
James Noble, Robert Biddle, Ewan Tempero, Alex Potanin, Dave Clarke

Encapsulation is a founding principle of object-oriented programming: to this end, there have been a number of recent proposals to increase programming languages with support for stronger encapsulation disciplines. While many of these proposals are similar in concept, it is often difficult to describe their effects in practice, or to evaluate clearly how related proposals differ from each other. To address this issue, the authors are developing a general topological model of encapsulation for object-oriented languages based on a program's object graph. Using this model, the authors can characterise a range of confinement, ownership, and alias protection schemes in terms of their underlying encapsulation function. This analysis should help programmers understand the encapsulation provided by programming languages, assist students to better compare and contrast the features of different languages, and help language designers to craft the encapsulation schemes of forthcoming programming languages.

Comments. This work is rather preliminary and the big question is where should it be taken. Answering this resulting in some quite lively discussion which then diverged (see later). Some audience members wanted the authors' work to allow one to compare the properties enabled by different encapsulation schemes. Other audience members wondered whether the framework could be used to give sufficient conditions for achieving modular soundness.

Lightweight Confinement for Featherweight Java
Tian Zhao, Jens Palsberg, Jan Vitek

Confinement properties impose a structure on object graphs which can be used to enforce encapsulation. This is essential for a number of reasons: to enable certain program optimizations, to reason modularly, and for software assurance. The paper formalizes the notion of *confined type* [VB01] in the context of Featherweight Java. The static type system which mirrors the informal rules of the original Confined Types proposal enables the system to be proven sound.

Comments. This work reflects one end of the spectrum of possible confinement schemes. The aim is to enforce a simple security property in a manner which has as minimal impact as possible, rather than providing as much flexibility as possible. In particular, there is the requirement that no extension to Java is required — a few additional checks are permitted.

A Static Capability Tracking System
Scott F. Smith, Mark Thober

Capabilities may be used within programming languages to define and enforce security policies: a simple example is that a user must hold a reference to an object to be able to use its features. The authors define a system for statically tracking capability flow via types. This work is related to Bokowski and Vitek's Confined Types [VB01] and to Skalka and Smith's `pop` system [SS02]. These previous systems concentrated on constraining capability flow at an object level, and enforcement at object and package points. The system outlined in this paper was a general means for confining the flow of capabilities in a program. This was done using a toy functional language, *fcap*, which included a general mechanism for declaring and enforcing secure boundaries for capability flow. The authors show how these boundaries may be statically enforced through typing. A sketch of how Confined Types and `pop` may be encoded into *fcap* was also shown.

Comments. A simple unifying calculus such as this could provide a useful basis for studying confinement schemes without all the complexity required to handle programming languages like Java. On the other hand, inheritance and subtyping make problems more acute in the object-oriented setting.

4 Discussion

Plenty of time was reserved for discussion, during which quite a few participants willingly played Devil's advocate. What follows is a synthesis of that discussion.

4.1 Controversy

Manual Fähndrich made the following comment which, for a workshop dedicated in part to aliasing, is rather controversial:

> We couldn't care less about aliasing.

This point is significant. Although aliasing is an immediate consquence of one of the defining characteristics of object-oriented programming (the ability to refer to an object using its identity), Manual believes that research should focus on what properties we want to use when reasoning about objects, rather than concentrating on aliasing. The properties come first and only when required must aliasing issues be considered. Only then should one delve into the ever-increasing bag of tricks for addressing alias problems.

4.2 Recurring Themes

A number of issues reccurred throughout the workshop, often under various guises. These, often overlapping, issues can be characterised as:

1. transfer of ownership
2. re-entrancy and recursion
3. the power-to-weight ratio of alias control disciplines

1. Transfer of Ownership. Two notions of ownership are being pushed around at the moment. Both aim to localise the access to objects, restricting which objects can modify an object — called *the owner*. In both cases, the most challenging problem is that of *ownership transfer*. How can we safely change the owner of an object to another owner?

This first form of ownership imposes structural invariants on object graphs, ultimately controlling which objects can access which other objects. This form of ownership is realized in *ownership types* [CPN98] and *confined types* [VB01]. Such systems offer statically checkable constraints on object access, though often at the cost of many type annotations. These have been used (in various forms) for sound modular reasoning about programs [MPH99], deadlock and race-free Java programs [BR01,BLR02], safe lazy software upgrades in object-oriented databases [BLS02], memory management in real-time Java [BSBR03], program understanding in the context of Java extended with architectural description constructs [AKC02,ACN02], and in demonstrating representation independence for Java [BN02]. Unfortunately, ownership types are unable to deal with cases which rely on more subtle invariants.

The second form of ownership is more subtle. Two or more objects or threads may have access to an object or some other shared resource, and ownership corresponds to having privileged access to modify or otherwise use a resource. The conditions under which ownership holds may depend upon subtle invariants of program state. Simple examples include whether a particular semaphore is held or whether a monitor has been entered, but much more complex variants are possible. Since the privileged access often depends upon perspective, Peter O'Hearn [O'H02] describes this form as "ownership is in the eye of the asserter." We dub it *owner-as-privilege*. Although generally not statically checkable, this approach permits privileged access to objects even in the presence of aliasing (though not necessarily in the presence of threads, as the discussion above suggests).

The two forms of transfer of ownership required for these types of ownership can thus be seen as transfer of pointer and transfer of privilege.

In the first case, a number of requirements exists to ensure that the transfer occurs correctly. Since objects are transferred from the representation of one object to another, we require that there are no residual pointers to the transferred objects from the original owner, nor that there are any references from the transferred objects to objects within the original representation. External Uniqueness of Clarke and Wrigstad [CW03] goes some way towards enabling this form of ownership transfer, but it cannot generally handle when fragments of data structures are ripped out and moved from one object to another.

In the second case, the privilege must implicitly be passed by changing the focus of the asserter. Ownership transfer often occurs via some protocol (e.g., obtaining a semaphore), though it may be the case that no actual operation corresponds to the ownership transfer. The program may simply enter a state in which, if everything is correct, the privilege safely and implicitly transfers. As in the first case, this requires that the original owner no longer acts as the owner, and also that no additional objects come to consider themselves as owner, apart

from the object in the asserter's eye. In the presence of subtyping and information hiding, this may be very difficult to achieve.

The question is whether we can develop lightweight static analyses for determining when these forms of ownership transfer are possible.

2. Re-entrancy and recursion. A recurring challenge is reasoning in the presence of re-entrancy and recursion. Re-entrancy occurs whenever a thread (which may be the only thread) which has been in an object (having called one of its methods), leaves the object and then, potentially via an alias, enters the object again. Essentially, the problem is that the properties which hold before entering an object and upon re-entering an object are generally and often intentionally different. It is not required, for example, that a class invariant hold during a method call. With recursion, the precondition required for the initial call must be re-established for every recursive call. Once invariants become complex, such as those involving separation logic or presenting complex resource demands, it becomes difficult to formulate single invariants which captures the requirements of both the initial and subsequent recursive calls. So how then does one reason modularly about method calls in the presence of re-entrancy? How does one devise sound and complete proof rules, especially for more demanding logics?

To make the problem a little clearer, consider a problem posed by David Naumann. Again we consider the difficulty of transferring the representation from one object to another, ensuring that no reference from the original object to the representation remains. This problem becomes more difficult when one considers that the method doing the move may be a re-entrant method into the object. This could mean that there may be references to the object(s) being transferred on a prior stack frame. When that earlier frame is re-invoked, the pointers which remain could be manipulated, thus manipulating the representation of the object to which the things were moved. The question is how do we deal with this? How does the transfer method know which pointers are present on the stack? How does the earlier method know which pointers may potentially move in subsequent method calls? How can we express these concerns in a general and useful manner?

The obvious solution, to re-establish class invariants before calling any other method, is clearly not practical. In their promising approach, Barnett, DeLine, Fähndrich, Leino and Schulte [BDF$^+$03] address this problem using the notion of *owner exclusion* to track who can access objects, allowing only one to access the representation at a time. It is not yet clear how practical this approach is.

3. Power-to-weight ratio of alias disciplines. A number of aliasing disciplines exist. Some have a rather large cost often in terms of syntactic overhead, but they are flexible and deliver strong aliasing guarantees. Other aliasing disciplines have little syntactic or static checking overhead, but also offer very few guarantees. Yet others enable strong reasoning principles, though are not applicable to all programs written in an conventional object-oriented style. Thus aliasing

disciplines differ in the power they provide (reasoning principles bought, theorems attainable, safety/security achievable, presence of other useful properties, e.g., modular soundness) and their weight (restrictions on programming style, usability, practicability, applicability, volume of annotations, cost of checking, complexity, expressiveness). The selection of an appropriate aliasing discipline becomes an issue.

The first answer to this question comes by first returning to Manuel's controversial remark. One must start with the property one wishes to enforce and then select the discipline which enables that to be (most easily) enforced.

Given that there is such a large variety of confinement schemes offering different properties, a more sensible long term solution would be to place the entire discipline under programmer control. The specification of the confinement discipline could be done within a flexible framework and called a *static-analysis aspect.* This observation was perhaps most entertainingly captured in David Naumann and Anindya Banerjee's motto:

"What do you want to encapsulate today?™"

4.3 Other Issues

Interestingly, approaches to the problems considered such as read-only, argument independence and so forth were not discussed. We offer two reasons why this may have been the case. Firstly, these approaches are conceptually simple and already well understood, and hence raise no further pressing questions. Secondly, it may be the case that these approaches are too restrictive and we are now getting to the stage where we are able to offer sensible solutions without being so restrictive. What seems to be most important now is enforcing/maintaining some sort of privileged access to objects.

5 Future

Due to its popularity, we plan to repeat the workshop, if not in 2004, certainly in two years' time.

References

[ACN02] Jonathan Aldrich, Craig Chambers, and David Notkin. ArchJava: Connecting software architecture to implementation. In *ICSE*, May 2002.

[AKC02] Jonathan Aldrich, Valentin Kostadinov, and Craig Chambers. Alias annotations for program understanding. In *OOPSLA Proceedings*, November 2002.

[BDF+03] Mike Barnett, Robert DeLine, Manuel Fähndrich, K. Rustan M. Leino, and Wolfram Schulte. Verification of object-oriented programs with invariants. In *Formal Techniques for Java-like Programs*, July 2003.

[BLR02] Chandrasekhar Boyapati, Robert Lee, and Martin Rinard. Ownership types for safe programming: Preventing data races and deadlocks. In *OOPSLA Proceedings*, November 2002.

[BLS02] Chandrasekhar Boyapati, Barbara Liskov, and Liuba Shrira. Ownership types and safe lazy upgrades in object-oriented databases. Technical Report MIT-LCS-TR-858, Laboratory for Computer Science, MIT, July 2002.

[BN02] Anindya Banerjee and David A. Naumann. Representation independence, confinement, and access control. In *Proceedings of the 29th ACM Symposium on Principles of Programming Languages (POPL'02)*, Portland, Oregon, January 2002.

[BR01] Chandrasekhar Boyapati and Martin Rinard. A parameterized type system for race-free Java programs. In *OOPSLA Proceedings*, 2001.

[BSBR03] Chandrasekhar Boyapati, Alexandru Salcianu, William Beebee, and Martin Rinard. Ownership types for safe region-based memory management in real-time java. In *ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI)*, June 2003.

[Cla03] Dave Clarke, editor. *Proceedings of the First International Workshop on Aliasing, Confinement and Ownership in Object-oriented Programming (IWACO)*. Number UU-CS-2003-030. Utrecht University, July 2003.

[CPN98] David Clarke, John Potter, and James Noble. Ownership types for flexible alias protection. In *OOPSLA Proceedings*, 1998.

[CW03] David Clarke and Tobias Wrigstad. External uniqueness is unique enough. In *ECOOP Proceedings*, July 2003.

[FD02] Manuel Fähndrich and Robert DeLine. Adoption and focus: Practical linear types for imperative programming. In *Proceedings of the ACM Conference on Programming Language Design and Implementation*, June 2002.

[HLW+92] John Hogg, Doug Lea, Alan Wills, Dennis de Champeaux, and Richard Holt. The Geneva convention on the treatment of object aliasing. *OOPS Messenger*, 3(2), April 1992.

[IO01] Samin Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *28th ACM Symposium on Principles of Programming Languages*, January 2001.

[MPH99] P. Müller and A. Poetzsch-Heffter. Universes: A type system for controlling representation exposure. In A. Poetzsch-Heffter and J. Meyer, editors, *Programming Languages and Fundamentals of Programming*. Fernuniversität Hagen, 1999.

[O'H02] Peter O'Hearn. Notes on separation logic for shared-variable concurrency. January 2002.

[SS02] Christian Skalka and Scott Smith. Static use-based object confinement. In *Proceedings of the Foundations of Computer Security Workshop (FCS '02)*, Copenhagen, Denmark, July 2002.

[SY86] Robert E. Strom and Shaula Yemeni. Typestate: A progamming language concept for enhancing software reliability. *IEEE Transactions on Software Engineering*, SE-12(1):157–170, January 1986.

[VB01] J. Vitek and B. Bokowski. Confined types in Java. *Software Practice and Experience*, 31(6):507–532, 2001.

A People

Participants

Jonathan Aldrich	University of Washington (USA)
Jörg Bauer	Saarland University (Germany)
Richard Bornat	Middlesex University (UK)
Chandra Boyapati	MIT (USA)
John Boyland	University of Wisconsin-Milwaukee (USA)
Cristiano Calcagno	Imperial College, London (UK)
Aziem Chawdhary	Queen Mary, London (UK)
Dave Clarke	Utrecht University (The Netherlands)
Sophia Drossopoulou	Imperial College, London (UK)
Sari Eldahah	(Jordan)
Manuel Fähndrich	Microsoft Research (USA)
Stephen Freund	Williams College (USA)
Christian Gronthoff	Purdue University (USA)
Johannes Henkel	University of Colorado (USA)
Doug Lea	SUNY Oswego (USA)
Rustan Leino	Microsoft Research (USA)
Ivana Mijajlovic	Queen Mary, London (UK)
Nick Mitchell	IBM TJ Watson Research Labs (USA)
Peter Müller	Swiss Federal Institute of Technology (Switzerland)
David Naumann	Stevens Institute of Technology (USA)
Peter O'Hearn	Queen Mary, University of London (UK)
Richard Paige	York University (UK)
Arnd Poetzsch-Heffter	Kaiserslautern University (Germany)
Erik Poll	Nijmegen University (The Netherlands)
Noam Rinetzky	(Israel)
Yannis Smaragdakis	Georgia Tech. (USA)
Matthew Smith	Imperial College, London (UK)
Mark Thober	Johns Hopkins University (USA)
Noah Torp-Smith	The IT University of Copenhagen (Denmark)
Jan Vitek	Purdue University (USA)
Tobias Wrigstad	Stockholm University (Sweden)
Yoav Zibin	Technion – Israel Institute of Technology (Israel)

Organisers

Dave Clarke	Utrecht University
Sophia Drossopoulou	Imperial College, London
James Noble	Victoria University of Wellington

Program Committee

Jonathan Aldrich	University of Washington
Dave Clarke (chair)	Utrecht University
Doug Lea	SUNY Oswego
David Naumann	Stevens Institute of Technology
James Noble	Victoria University of Wellington
Peter O'Hearn	Queen Mary, University of London
Martin Rinard	MIT
Jan Vitek	Purdue University

Author Index

Angster, Erzsébet 130

Bauer, Markus 107
Bergin, Joseph 130
Beugnard, Antoine 17
Börstler, Jürgen 119
Bosch, Jan 34
Brito e Abreu, Fernando 92
Bruce, Kim B. 119
Bryce, Ciarán 86

Chitchyan, Ruzanna 154
Clarke, Dave 197
Clemente, Pedro J. 50
Czajkowski, Crzegorz 86

D'Hondt, Theo 143
Davis, Kei 11
Demeyer, Serge 72
Dony, Christophe 1
Drossopoulou, Sophia 197
Ducasse, Stéphane 72, 143

Eisenbach, Susan 62

Fernández, Alejandro 119
Fiege, Ludger 17
Filman, Robert 17, 190
Finkelstein, Anthony 179

Genssler, Thomas 107

Hannemann, Jan 154
Haupt, Michael 190
Hoffmann, Hans-Jürgen 30

Jul, Eric 17

Knudsen, Jørgen Lindskov 1

Lamerdorf, Winfried 179
Leavens, Gary T. 62
Leyman, Frank 179
Lujan, Sergio 50
Lumpe, Markus 107

Madsen, Ole-Lehrman 143
Mehner, Katharina 190
Mens, Kim 72
Meuter, Wolfgang De 143
Michiels, Isabel 119
Müller, Peter 62

Noble, James 197

Pashov, Ilian 165
Pérez, Miguel A. 50
Piattini, Mario 92
Piccinelli, Giacomo 179
Poels, Geert 92
Poetzsch-Heffter, Arnd 62
Poll, Erik 62

Rashid, Awais 154
Reiser, Hans 50
Riebisch, Matthias 165
Romanovsky, Alexander 1
Rysselberghe, Filip Van 72

Sadou, Salah 17
Sahraoui, Houari A. 92
Schneider, Jean-Guy 107
Schönhage, Bastiaan 107
Sipos, Marianna 130
Streitferdt, Detlef 165
Striegnitz, Jörg 11
Szyperski, Clemens 34

Trifu, Adrian 72
Tripathi, Anand 1

Vasa, Rajesh 72

Weck, Wolfgang 34
Weerawarana, Sanjiva 179